FEATURE-BASED PRODUCT MODELING IN A COLLABORATIVE ENVIRONMENT

YANG LEI

(B.ENG., Xi'an Jiao Tong University, P.R. China)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2010

Name: YANG Lei

Degree: Doctor of Philosophy

Dept: Mechanical Engineering

Thesis Tile: Feature-based Product Modeling in a Collaborative Environment

Abstract:

A replicated collaborative feature modeling system has been explored in this study, where a team of designers work together creating prismatic product models or designing displacement features on freeform surfaces. Two modeling functions are enhanced in this work, namely a history-independent modeling approach used for regular feature modeling and a surface blending approach used for displacement feature modeling. In addition, a granular locking mechanism has been explored for scheduling the concurrent design operations at the server. In this modeling system, users can perform design operations on a product model concurrently, e.g., create and modify regular-shaped features, designing some intricate features on freeform surfaces, and the server coordinates the concurrent operations and synchronizes the product information. This modeling platform provides a valuable paradigm for designers working together on a complex product model, which is strongly needed in current product development.

Keywords: boundary evaluation; collaborative; feature; granular locking; product modeling; surface blending

Pursuing a PhD is really an enduring and dedicated task, and it cannot be finished without the support, guidance, and encouragement from many people.

First of all, I would like to express my great gratitude to my supervisors: Professor Andrew Nee Yeh Ching and Associate Professor Ong Soh Khim. Without their guidance and patience, I cannot finish this PhD work in the past four years. I always remember the beginning days when I first came to NUS. At that time, my English was poor and I did not have much sense of PhD research. Their patience and inclusiveness encouraged me to learn and progress. I did have had much stress during my PhD study, but it is the stress that has propelled me to learn more and finish the research work on time.

I would also thank the most important people in my life: my families. They always show great encouragement and support for my study in Singapore, and they always prove that I have a safe port to dock if I really feel tired.

I am grateful to the research students and researchers in our lab: Lou Ping, Shen Yan, Li Jun, Niu Sihong, et al. Thanks for their encouragement and discussion of my research project. They shared many ideas and their encouragement released my mind when I felt helpless.

At last, I would like to thank many friends in NUS: Li Erqiang, Wang Shouhua, Zhang Bao, Lin Yingshuai, Liu Gang, et al. They are kind-hearted and excellent in both academic and daily life aspects. We played together and talked freely, which brought much fun to our monotonous study life.

Table of Contents

Summary

Computer-aided product modeling has been a research topic since its advent in product development. Many modeling techniques have been employed in the past few decades, e.g., feature-based design, freeform surface modeling, collaborative feature modeling, etc. However, exploring and enhancing the modeling functions remains as a research topic for improving design quality and shortening development time, especially for concurrent and collaborative product design. In this study, a replicated collaborative feature modeling framework has been proposed and validated, in which the designers can work together creating a prismatic model and designing displacement features on freeform surfaces.

At the client sides, each user is provided with the full-fledged modeling functions, in which two modeling functions have been enhanced in this work. Firstly, a history-independent modeling approach has been proposed and validated for overcoming the problems and shortcomings in current history-based modeling. In this approach, when a feature is modified, it is first removed from the product model by updating its intersecting features, and it is then re-added with the newly specified parameters. Hence, the creation step of the feature being modified is changed, and the problems caused by the static ‘feature creation order’ can be solved. The complexity analysis and performance measurement of the proposed boundary evaluation algorithm for three representative models show that its computational complexity is better than history-based modeling. Secondly, to avoid the high polynomial degree of the tangent field curve obtained symbolically, an approximation for the Cubic Hermite Interpolant has been proposed and validated. The boundary curve of the displacement feature is

first offset in the tangent field with a user-specified tolerance, and it is then knot-refined to be compatible with the offset curve for surface blending. The local self-intersection problem in the offset curve is eliminated in the parametric space by approximately mapping the offset vectors in the respective tangent planes to the parameter space of the base surface. The examples studied using the proposed algorithm show that the boundary curve of the displacement feature can be specified flexibly by the users, and the normal deviation along the boundary curve is even smaller than the offset tolerance.

At the server side, a granular locking mechanism is employed for scheduling the concurrent design operations and resolving potential operation conflicts. The design operations are grouped according to feature dependency relationships, so more than one ‘modify operation’ can be executed concurrently as long as their dependency scopes are mutually exclusive. The potential conflicts of design operations caused by feature interactions have been resolved using a naming and matching mechanism, through which the correspondence of the modified topological entities would be achieved correctly.

List of Tables

List of Figures

Chapter 1 Introduction

Product modeling is a process of defining a computer-aided design (CAD) model or its explicit representation that satisfies the functional requirements expected by the users (Shen *et al*., 2001). According to the designed CAD model, the machining process is generated and executed on computer numerical controlled (CNC) machines to produce the required workpiece. In the beginning era of computer-aided design, geometric modeling was developed to facilitate designers to create and manipulate the CAD models, which can be represented as graphical models, solid models and surface models (Shah and Mäntylä, 1995). However, geometric modeling has some deficiencies, such as the lack of design intent, tedious modeling procedure, etc. In order to overcome the limitations of geometric modeling, some semantic and high-level entities are required to represent the CAD models. With this consideration, feature modeling has emerged as a promising solution, where product modeling is a process of combining certain specific features into a stock model; thus feature modeling provides a high-level and efficient modeling environment (Roller, 1989; Shah, 1991). Furthermore, engineering specifications attached to the features enable seamless connection between different domains in the product development cycle, which has the benefit of reducing lead-time and improving product quality. Nevertheless, the majority of the current feature-based design systems are history-based modeling, which has some weaknesses and shortcomings. In addition, freeform features, which are popularly used in aesthetic design and engineering product design, are not supported in current feature-based design systems. The shortcomings and limitations of current feature-based design need to be addressed in order to employ it effectively in product development.

Besides the development of high-level modeling environments, outsourcing has become a significant trend in the current global manufacturing market, especially for large firms, such as Boeing, Ford, Kodak, etc. Under this scenario, product design has been shifted from standalone to collaborative activities (Li *et al.*, 2004; Wang and Nee, 2008). As such, adopting feature-based design in a collaborative environment has become a topic of research. In collaborative computer-aided design, a team of experts work together on product design (Ding *et al*., 2009; El-Tayeh *et al*., 2008), so a coordination mechanism is strongly needed for scheduling the concurrent design activities and managing the operation conflicts.

The subsequent sections provide an overview of feature-based design and collaborative computer-aided design. A more detailed discussion of the reported research works in the relevant areas will be presented in Chapter 2.

1.1 Feature-based Design

The feature-based modeling technique has been widely used in both commercial and academic computer-aided X (CAX) systems; it provides an effective approach for improving design efficiency and assisting product model translation across different domains. In feature-based design, the product model is created by combining certain specific features, each of which is defined as a parametric shape associated with certain functional information and constraints (Bidarra and Bronsvoort, 2000; Sheu and Lin, 1993; Wang and Nnaji, 2006). From the CAD models, manufacturing features are recognized (Lee and Kim, 1998; Li *et al*., 2001; Rahmani and Arezoo, 2006) for automating the machining process on CNC machines. Furthermore, by combining feature-based design and feature recognition (Duan *et al*., 1993; Laakko and Mäntylä,

1991; Martino *et al*., 1994), the design flaws in a CAD model can be investigated immediately, such that the design quality can be guaranteed. Analogously, other downstream application processes can extract a specific feature model from a CAD model, so the geometric reasoning in the specific domains can be automated. The feature models extracted in different domains can be converted from one to another (Bronsvoort and Noort, 2004; Hoffmann and Joan-Arinyo, 2000; Subramani and Gurumoorthy, 2004), such that the CAD system can be integrated seamlessly with the subsequent applications, e.g., manufacturability analysis, process planning, etc.

Although feature-based design has been widely used in product development, it still has some weaknesses and shortcomings that are only partly resolved in the literature (Bidarra and Bronsvoort, 2000), e.g., the feature model is usually a macro that is only supported in the design interface, and lacks the persistent maintenance of feature validities, etc. More importantly, the majority of the current feature-based design systems is history-based, where all the 'feature creation operations' are stored in the model history and they are static. After each modification, the model history is sequentially re-executed to update the resulting boundary representation (B-rep). This evaluation mechanism causes some problems, e.g., the evaluated model does not correspond to its specification, the operation can only refer to the boundary entities created by the previous operations, high computation cost, etc. For solving the problems caused by the static 'feature creation order', a cellular representation and modeling scheme was reported by Bidarra and Bronsvoort (2000), where the non-associative set operations (union and difference) were replaced by a non-regular union operation. For overcoming the high-computation cost, two methods have been devised and developed, namely, storing all the intermediate B-rep models at each history step,

and storing only the deltas between the history steps (Bidarra *et al*., 2005). However, these proposed approaches cannot solve the problems in current history-based feature modeling effectively.

In addition, current feature-based design does not support freeform surface modeling, which is increasingly needed in aesthetic design and product design. As reported by Cavendish and Marin (1992), embedding a number of displacement features into a base surface is popular in industrial product design and modeling. By using the feature modeling technique, the freeform surface can be created and modified intuitively, since some intuitive and user-friendly parameters can be associated with the underlying mathematical model (Nyirenda and Bronsvoort, 2009; Pernot *et al*., 2008; van den Berg *et al*., 2002). Under this consideration, displacement feature modeling has been explored in the literature (van Elsas and Vergeest, 1998), and it has two important modeling steps, namely, specification of a boundary curve on the base surface and surface blending of two non-interacting surfaces. In surface blending, the Cubic Hermite Interpolant is usually adopted for achieving the tangent plane smoothness across the boundary curve (Elber, 2005; van Elsas and Vergeest, 1998). Whereas, in this situation, the polynomial degree of the tangent field curve obtained symbolically is considerably higher, and the degree of reduction of a freeform curve is a non-trivial task. As a result, an effective surface blending approach is needed for achieving the smoothness across the boundary curve.

In summary, the feature modeling technique has many advantages in product design and manufacture, and it has been a topic of research effort in the past few decades. However, the weaknesses and limitations in current feature-based design remain as an

obstacle hindering its effective application. Hence, further research effort is still necessary for improving the usability of feature-based design in product development.

1.2 Collaborative Computer-aided Design

Global manufacturing market and competition have been driving companies to deploy new ‘product development' paradigms for improving product quality and shortening lead-time. Under this situation, it is well realized that the paradigm of product development is moving towards engaging and coordinating different application domains, which forms a collaborative and distributed development environment based on the distributed software modules and information technology, e.g., CORBA, Java RMI, Agent, and COM etc.

In collaborative design, groups of experts work together on product design (Ding *et al*., 2009; El-Tayeh *et al*., 2008; Rosenman and Wang, 1999), so as to identify and resolve design problems at an earlier stage of the product life-cycle. The collaboration was categorized into three types by Li and Qiu (2006), namely, visualization-based collaboration for conceptual design and product review, cooperative creation and manipulation (co-design) for detailed design, and concurrent engineering integrating the design and the related manufacturing processes. The co-design system has two widely used architectures, namely, centralized system where the main modeling functions are located at the server side, and replicated system where each designer is provided with the full-fledged modeling capabilities. In this study, only the replicated system is focused to explore a platform for collaborative feature modeling.

In a co-design environment, a part model is co-created and co-manipulated by a team of designers so as to improve design quality and design efficiency. In this situation, a

coordination mechanism for scheduling the collaborative design activities and managing operation conflicts is crucial (Li *et al*., 2008b), since a team of designers intends to create and manipulate the part model at the same time. In the literature, the locking mechanism is usually adopted as the coordination scheme, either a total locking mechanism (Bidarra *et al*. 2002; Li *et al*. 2004; Li *et al*. 2007) or a granular locking mechanism (Chan and Ng 2002; Li *et al*. 2008b). By the total locking mechanism, only the designer who holds the control baton can edit the design model, but other co-designers only observe or comment on the design operation and receive the updated model information. By the granular locking mechanism, the locking granularity is finer that the design model is divided into several portions, thus more than one designer can edit different portions at the same time. However, there are some limitations of the currently reported locking mechanisms. By the total locking mechanism, since the control baton is permitted to one designer at one time, the design model is edited by the designers in a sequential order. This is not a productive collaboration mechanism, although the collaboration can be manifested such that all designers can review and discuss a design operation together before its execution (Shen *et al*. 2006). By the granular locking mechanism, since performing the design operations concurrently may cause operation conflicts and model inconsistency, the definition of the locking granularity and the potential conflict resolution is critical. As a result, the coordination mechanism needs more research effort for employing it effectively in a co-modeling environment.

In summary, distributed and collaborative design has been popularly investigated and employed in product development for improving design quality and shortening product time-to-market. However, the challenges and problems would need to be further

considered and addressed for establishing an integrated and collaborative environment, especially an effective coordination mechanism for scheduling the concurrent design activities. As a result, collaborative computer-aided design remains as an open research area, and it needs further investigation.

1.3 Motivations and Research Objectives

Research gaps for the current study of product modeling in a collaborative environment are summarized below:

- History-based feature modeling has some shortcomings due to the static 'feature creation order', such as generating undesirable product model, restricting reference entities, high computation cost, etc.
- In collaborative feature modeling, some issues need to be addressed for employing granular locking mechanism, such as maintaining exclusive 'feature creation order' and resolving operation conflicts.
- Currently, there are few studies on adapting freeform feature modeling in a collaborative design environment.

The overall aim of this study is to provide a design platform for creating product models collaboratively and concurrently, with either regular prismatic models or displacement features on a freeform surface. The investigated system is basically for replicated collaborative feature modeling, as shown in Fig. 1.1. At the client sides, each designer is provided with the full-fledged modeling functions for both regular features and freeform surface features, and the server coordinates the design activities and synchronizes the product information. The specific objectives of this research are:

1) Aim I: Propose a history-independent modeling approach to overcome the shortcomings in current feature-based modeling, in which the creation sequence of

the features can be changed after each ‘modify operation’. The proposed modeling approach may provide insights into the boundary evaluation in feature-based modeling. However, it should be noted that the structure of a feature-based system is very complex, and the entire structure is not the central point of this study. The focus here is the modeling procedure of its boundary evaluation.

2) Aim II: Improve the granular locking mechanism for replicated collaborative feature modeling, in which the operation conflicts are resolved using a naming and matching mechanism. The proposed conflict resolution mechanism should be a valuable supplement for the granular locking mechanism. It should be noted that the proposed locking mechanism is only used in replicated co-design system, and for prismatic product modeling.
3) Aim III: Propose a freeform feature modeling approach for creating displacement features on freeform surfaces, and adapt this modeling approach in a collaborative environment. More specifically, a surface blending approach for generating the transition surface in displacement features with tangential smoothness across the boundary curve was investigated. The smoothness across the boundaries can be specified intuitively by setting the radius parameters, and the shape of the transition surface can be controlled by setting its control points. This work may shed light on creating displacement features in an efficient and intuitive process. There are many issues involved in freeform feature modeling, such as 3D curve mapping, boundary curve specification, degree reduction of freeform curves, etc., which are only discussed briefly in this work. The focus here is the surface blending for the transition surface.

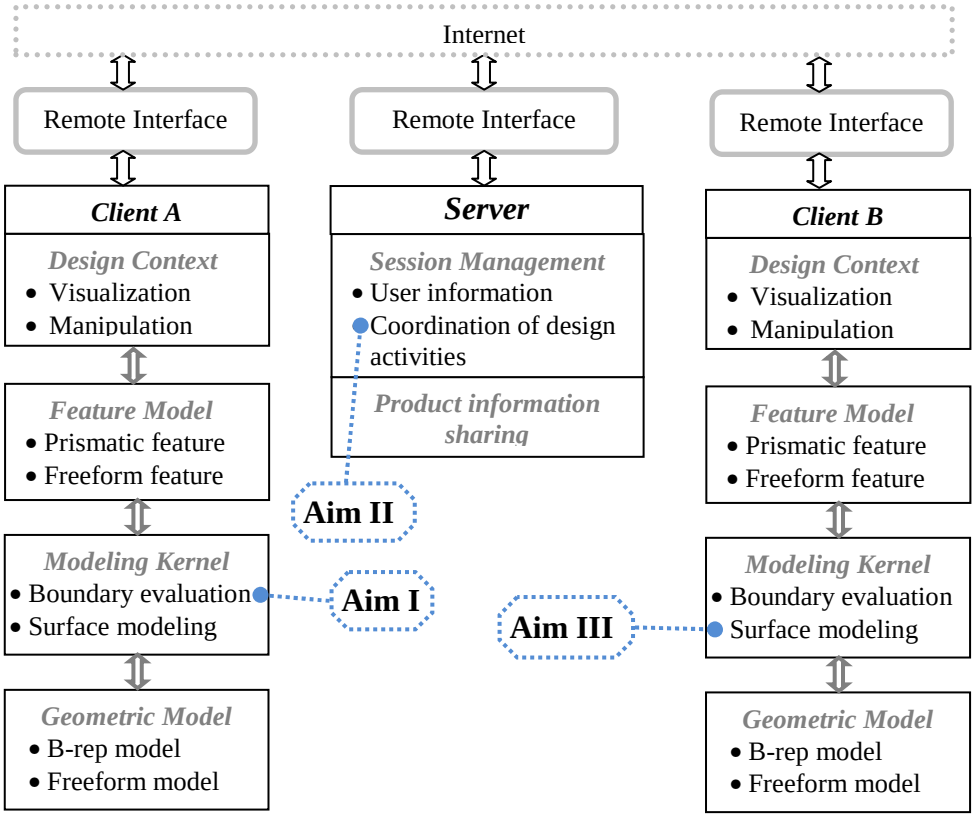


Fig. 1.1 System framework

1.4 Outline of Thesis

The remaining sections of this thesis are organized as follows:

In Chapter 2, the reported works in feature-based design and collaborative computer-aided design are surveyed and discussed. More specifically, the relevant works within the research objectives are investigated and discussed in detail, namely, boundary evaluation in feature-based design, coordination mechanism in collaborative feature modeling, and displacement feature modeling.

In Chapter 3, a history-independent modeling approach is presented. The weaknesses and shortcomings in the current feature-based design systems are overcome. The working principle and the advantage of the proposed modeling approach are presented, and the computational complexity is investigated and compared.

In Chapter 4, a granular locking mechanism for replicated collaborative feature modeling is presented. The resolution of operation conflicts and the consistency maintenance of ‘feature creation order’ are elaborated.

In Chapter 5, freeform feature modeling and its adaption in a collaborative environment are presented. The two issues in displacement feature modeling, namely, specification of feature boundary and surface blending, are elaborated in detail. For its application in a collaborative environment, the coordination and product information sharing are discussed briefly.

In Chapter 6, the implementation tools and methods used in this study are presented, in which the software modules and the programming environment are discussed. The structure of the proposed collaborative design system is shown, and the two types of product models that can be used in this modeling system are presented.

Finally, Chapter 7 concludes this thesis, in which the contributions of this research work and the suggestions for future work are presented.

Chapter 2 Literature Review

This chapter presents a survey of the literature pertinent to the studies on feature-based design and collaborative computer-aided design. Firstly, the feature modeling technique used in product design and modeling is investigated. The studies on regular feature modeling that is used for the design of prismatic parts are reviewed and discussed, and the corresponding feature-based design system is investigated. In addition, the studies on freeform feature modeling and modification are surveyed, and the applications of freeform features and its modeling procedure are presented. Secondly, the pertinent studies on collaborative computer-aided design are investigated, where the coordination mechanism used for scheduling the concurrent design activities and the synchronization mechanism used for product information sharing are reviewed in detail.

2.1 Feature Modeling Technology

The feature modeling technique has been popularly used in product development, including product design, manufacturability analysis, process planning, etc. In addition, freeform feature modeling is proposed for creating and manipulating freeform shapes intuitively, which are widely used in aesthetic and engineering design. In this subsection, the applications of feature modeling in product development are reviewed, including design-by-feature, feature reorganization, and multiple-view feature modeling. It is followed by the investigation of current feature-based design system, in which two issues are highlighted, namely, problems caused by the history-based modeling procedure, and the persistent naming problem. Finally, the applications of freeform feature modeling and the relevant studies are reviewed. Specifically, the

modeling procedure and the relevant issues of displacement feature modeling are highlighted.

2.1.1 Feature Modeling in Product Development

In this subsection, the studies on feature specification, feature modeling in product development, and multiple-view feature modeling are reviewed for an in-depth understanding of the feature modeling technique.

2.1.1.1 Feature Specification

A feature can be formalized in two approaches, namely, procedural formalism in which a feature is defined in terms of rules and procedures, and declarative formalism in which a feature is defined in terms of sets of constraints. The general specification of a feature involves the following information:

1) Geometry definition of the feature shape: each feature shape is a specific part of the resulting geometric model. Its geometric representation can be described using four structures (Shah, 1991), namely, augmented graphs, algebraic (syntactic), delta volumes, and constraint-based B-rep, all of which specify the spatial relationships of the geometric entities that constitute the feature.
2) Validity condition: it is the functional requirements of a feature, which may be violated due to feature intersections. As suggested by Bidarra and Bronsvoort (2000), feature validity can be represented as the topological constraints on the feature faces, which need to be maintained during the design process.
3) Annotation: it is the deposited information on the feature entities, such as tolerance, machining condition, etc. It does not change the feature shape and the validity, and can be updated automatically along with the topological modifications, as reported by Hoffman and Joan-Arinyo (1998a, 2000).

Keeping the three aspects in consideration, a few feature definition and representation approaches have been reported in the literature. Duan *et al.* (1993) reported a procedural approach, in which a feature is defined as a parametric-shape unit, consisting of a geometric description, attributes, and application-oriented mapping methods for design and manufacturing purposes. Laakko and Mäntylä (1993) reported a feature definition frame, which contains topology-definition, geometry-definition, auxiliary-geometry entities, geometric constraints, rules and attributes. A form feature representation was reported by Sheu and Lin (1993), in which each feature is basically a solid primitive associated with certain measured entities, dimensions, locations and constraints. In the above approaches, each feature is simply defined as a solid shape using the common techniques, e.g., primitive instancing, sweeping, etc., and the solid shape is associated with certain high-level information and constraints. This approach provides an effective way to create and manipulate the part model by performing operations on the solid primitives. However, the feature model is only a macro supported in the design interface, and the underlying geometric model is not represented in terms of features. In addition, constraints associated with the solid primitives are not maintained during the design process, which may be violated due to feature interactions. In order to overcome this weakness, Bidarra and Bronsvoort (2000) reported a declarative feature modeling approach, in which each feature consists of a feature shape, validity conditions, and the user interface. The feature shape is defined by setting certain spatial constraints on the constitutive geometric entities, the parameter and validity conditions are also defined as constraints. This approach is useful in that the validity conditions of the feature model are maintained during the design process, since all the constraints are checked after each modeling operation.

In addition, since the predefined features are limited and domain dependent, Hoffmann and Joan-Arinyo (1998b) suggested an approach for creating user-defined features (UDF) from standard features. A UDF feature is a parametric shape consisting of a set of standard features, a set of constraints, a set of attributes, and a user interface. This approach is significant in that the specific features can be defined dynamically, since a universal set of features is almost impossible to be set up.

2.1.1.2 Feature Models in Product Development

Generally, a feature model can be created in two ways, namely, design-by-feature and feature recognition. In design-by-feature, the designers use a set of predefined features for constructing a product model by a sequence of feature attachment operations. The feature model is usually represented as a graph structure, which comprises of the features and the relationships between features. As in the Feature Dependency Graph (FDG) reported by Sheu and Lin (1993), it consists of the specific form features and the feature-position operator (FPO). The FPO represents the relative positioning relationship between two features, through which all the features can be combined quite easily together. A similar FDG was reported by Bidarra and Bronsvoort (2000) for representing the feature model, which contains all the feature instances and their interacting constraints. In design-by-feature, a feature model can be created easily, which is from the design perspective. However, the feature models used in design and manufacturing are defined and perceived in two different perspectives, thus manufacturing features need to be recognized from a designed CAD model during feature-based machining. In feature recognition, the machining process of the part model is recognized, and is represented as a set of specific features, which can be used for process planning later. As suggested by Shah (1991, 1995), feature recognition compares geometric entities with predefined generic features to identify instances that

match the predefined ones, which can be boundary-based and volume-based. The boundary-based method finds sets of faces that satisfy a set of conditions for each feature, including rule-based, graph-based, syntactic methods, whilst the volume-based method operates directly on constructive solid models, such as CSG trees.

Studies in feature recognition have been reported in the literature. Lee and Kim (1998) proposed an incremental feature recognition approach from a feature-design model. It can convert various design features, including depression features, transition features, and protrusion features, into machining features incrementally. The proposed mechanism takes three steps: firstly, the interacting volumes of an incrementally added design feature and the previous extracted machining features are checked; secondly, the added design feature and the interacting volumes are handled for the conversion into machining features using feature information, nominal geometry, and feature interaction; and lastly, the feasibility of the extracting machining features is analyzed. Likewise, Li *et al*. (2001) proposed a mechanism to extract manufacturing features from a design-by-feature model. There are three steps in this recognition mechanism. Firstly, the design feature tree is converted to an intermediate manufacturing feature tree (MFT). The essential point in this step is to identify the interacting relationships between a design feature and the manufacturing features in an incrementally evolved intermediate MFT. Secondly, the features in the MFT are converted into several alternative interpretations based on three consecutive operations, namely, combination, decomposition, and (tool approach direction) TAD-led operations. Thirdly, a single interpolation of features in the MFT is selected for a specific workshop environment, which has the lowest machining cost. In the above two approaches, the critical point is to handle the interacting volumes of a newly added design feature and the extracted

machining features. The incremental recognition approach is highly significant in that the manufacturing implications of design actions can be fed back instantly, so the design quality is guaranteed. Rahmani and Arezoo (2006) presented a hybrid graph-based and hint-based technique to extract interacting features automatically from solid models. The hint-based approach is used to find traces left by the motion of a milling cutter in the part boundary. The feature hints, which are simple graphs carrying information about a feature's base and side faces, are extracted from the decomposed graph of an Attributed Adjacency Graph (AAG) for a part. After that, a complete feature volume is generated using three geometric completion algorithms, namely, Base-Completion, Profile-Completion and 3D-volume generation algorithms. This approach is noteworthy in that the available approaches can be combined so as to handle the drawbacks in existing recognition systems.

Combining design-by-feature and feature recognition is an effective solution for improving design quality, since the manufacturability of the part model can be checked immediately. Several modeling systems have been reported in this realm. Laakko and Mäntylä (1993) reported a hybrid framework of feature-based design and feature recognition. In their design environment, designers can manipulate interactively either the solid model or the feature model of the part, which provides much freedom for the users. Martino *et al*. (1994) developed a modeling system integrating design-by-feature with automatic feature recognition. An intermediate model is devised as the bridge between geometric models and context-based feature models. The hybrid framework connects product design and manufacturing seamlessly, thus the design quality is improved and the development time is shortened.

2.1.1.3 Multiple-View Feature Models

In the downstream application processes, the product model is reviewed and analyzed from different perspectives. Hence, a feature model used in the specific application needs to be extracted from the designed CAD model. In order to connect the feature models in different domains, multiple-view feature modeling has been carried out in the literature. Two types of feature conversion mechanisms have been proposed in multiple-view feature modeling, namely, one-way and multiple-way conversion. In one-way conversion, product shape can only be modified in the design view, and the modifications in other views are extracted from the evaluated B-rep model. In multiple-way conversion, product shape can be modified in any feature view, and product modifications can be propagated across multiple views automatically. Hoffmann and Joan-Arinyo (2000) presented a master model for maintaining consistency across multiple-view feature models. The master model is a single repository that contains all the relevant product databases. Each modification of one feature model is transmitted to the master model, and then other feature models are updated based on the updated master model. This approach is novel in that the product shape can be modified in other views using constraint reconciliation rather than in the design view only. Jha and Gurumoorthy (2000) presented an algorithm to propagate feature modification automatically across different domains. The input of this algorithm is all the feature interpretations of a part, and the feature modification is restricted to feature geometry only. This algorithm is on the basis that the history/log of the feature extraction process has been obtained and used as the input. The limitation of this algorithm is that the modification is restricted to feature geometry only, which is not useful in many applications. This mechanism was extended by Subramani and Gurumoorthy (2004), which handles various feature modifications like

feature deletion, feature creation, transformation and parameter changes. There are two steps here, in the first step, the feature volumes in the target feature model are updated to account for the modifications in the edit-view, which are determined by the interaction between the feature volumes in the target-view and the edit-feature volume; in the second step, the updated feature volumes in the target-view are recognized to identify new features in the target feature model. Bronsvoort and Noort (2004) extended the multiple-view feature modeling to support four product development phases, namely conceptual design, assembly design, part detail design, and assembly design. In this approach, the feature models extraction and consistency maintenance are based on an intermediate cellular model. This approach has made a valuable contribution to multiple-view feature modeling, since it extends the feature models into conceptual design and assembly design.

In summary, feature modeling has been widely used in product design and manufacturing. A feature contains a parametric shape and the associated attributes that are used in downstream application processes, thus a feature model contains more information than a geometric model in that its geometric reasoning in specific applications can be automated. Through combining design-by-feature and feature recognition, the manufacturing implications of design actions can be fed back instantly so that design quality can be improved. Furthermore, multiple-view feature models in different domains can be connected and synchronized seamlessly for obtaining a concurrent working environment. The applications of feature modeling reviewed in this subsection provide a substantial understanding of feature-based design, and paves the way for the subsequent literature review in this Chapter.

2.1.2 Feature-based Design System

The majority of current design systems is feature-based modeling, which includes a model history and an evaluated geometric model. Feature-based design provides an attractive and high-level modeling environment, in which a part model is generated by combining some specific feature shapes. In this subsection, the system components in feature-based design are investigated and discussed. Specifically, the persistent naming problem and the boundary evaluation mechanism in current feature-based design are investigated.

2.1.2.1 Problems in Feature-based Design

The schema of current feature-based parametric modeling system is depicted in Fig. 2.1. In such a CAD system, a product model is represented in two separate layers, namely the parametric definition and the geometry description. The parametric definition is created based on predefined features, and is usually represented as a feature dependency graph that includes all the specified features and their dependency constraints. The resulting geometrical model is generated through evaluating the parametric definition using the boundary representation approach (B-rep). During the design process, the topological entities of the intermediate B-rep model are usually referred to in the new feature operations for attaching or positioning purposes, which are achieved through a naming scheme. During the re-evaluation of the model, the referred topological entities in the old B-rep model need to be mapped to the topological entities in the new B-rep model, which is achieved through a matching mechanism. Hence, a naming and matching scheme is usually used in feature-based modeling to assign an identifier to the referred topological entities, and map the identifier to the topological entities in the new B-rep model.

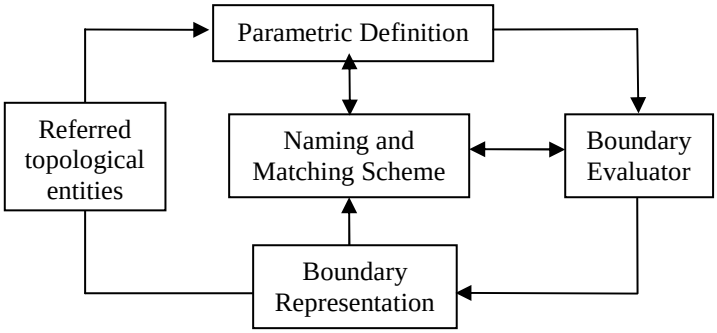


Fig. 2.1 Schema of the feature-based parametric model

The majority of current feature-based modeling systems is history-based, where all the 'feature creation operations' are stored in the model history. After each modification, the model history is sequentially re-executed to update the resulting B-rep model. During re-evaluation, a 'modify operation' is executed on the basis of the intermediate B-rep model that is generated by evaluating the previous operations in the model history. This evaluation mechanism causes some problems that have been reported by Bidarra and Bronsvoort (2000). The first problem is the reference entity problem, where a feature operation can only refer to the topological entities generated by the previous operations. As shown in Fig. 2.2, two features *BHole* and *Rib* are sequentially attached to an initial *Stock*. If a designer wants to modify and re-position the *BHole* relative to the *Rib* at a distance D, the positioning constraint cannot be defined since the *Rib* is created later than the *BHole*. The second problem is the model evaluation problem where the resulting B-rep model cannot be evaluated according to the designer's specification. As shown in Fig. 2.3, the designer can obtain the intended *THole* in (b) when the depth of the *THole* is equal to or larger than the height of the *Stock*, but he cannot modify the *THole* as the specification in (d) if the extruded *Block* is created later than the *THole*. From the designer's point of view, the modified *THole* would intersect with the *Block*. However, during the re-evaluation of the *THole* modification, the intermediate B-rep model at this step does not contain the

Block, so the *THole* only intersects with the *Stock* even the depth of the *THole* has been increased. As a consequence, the evaluated B-rep model is (e) which is not the intended model. In history-based modeling, the designer performs the 'modify operation' based on the current B-rep model, but the evaluation of the modified feature is on the basis of the intermediate B-rep model at its creation step. The difference between the current B-rep model and the intermediate B-rep model causes the above problems.

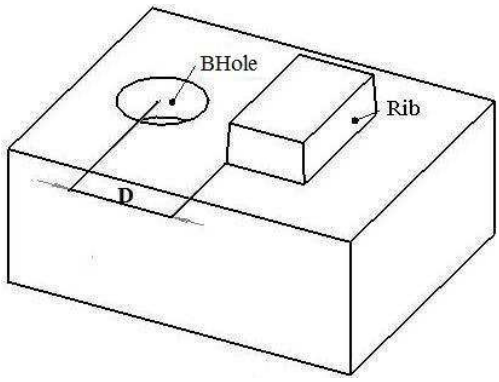


Fig. 2.2 Reference entity problem in history-based modeling

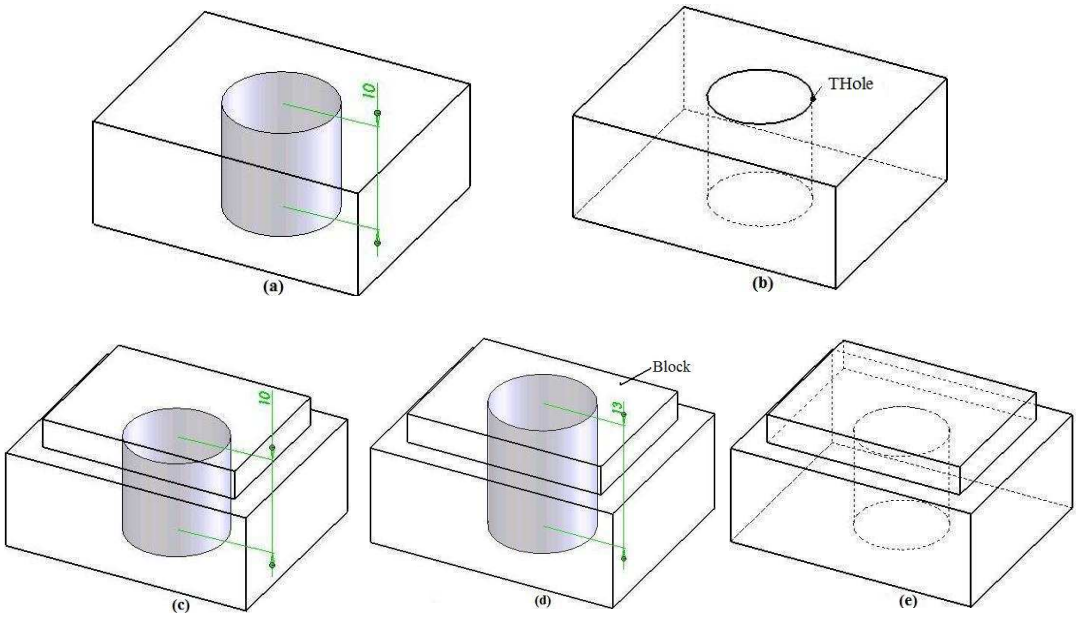
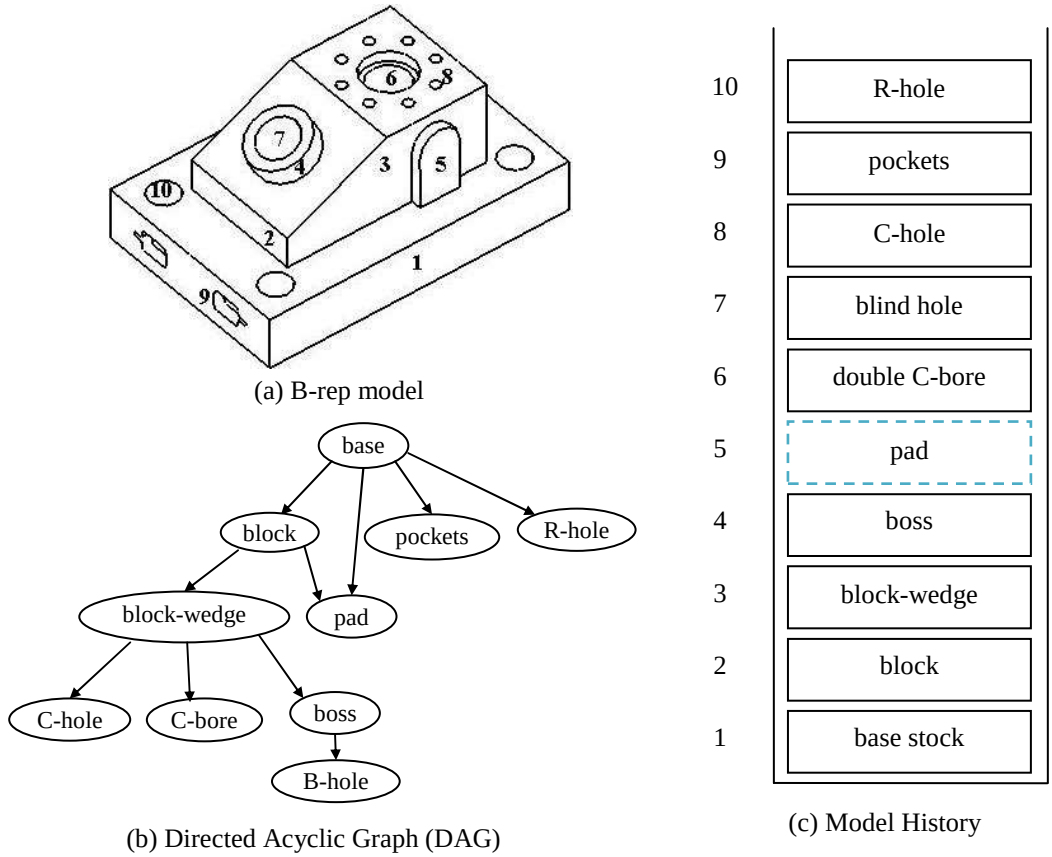


Fig. 2.3 Model evaluation problem in history-based modeling

High computation cost is another shortcoming in history-based modeling. After each modification, the entire model history needs to be re-executed, where the computation cost is proportional to the number of the features in the model history (Bidarra and Bronsvoort, 2000). This problem can be illustrated by the ANC 101 test part (Shah and Mäntylä, 1995) shown in Fig. 2.4, where (a) shows the resulting B-rep model, (b) shows the directed acyclic graph (DAG) of the design features, and (c) shows the model history. When the feature *Pad* is modified, the operations from step1 to step10 are re-executed to update the resulting B-rep model.



(a) B-rep model

(b) Directed Acyclic Graph (DAG)

(c) Model History

Fig. 2.4 CAMI-ANC 101 test part (Shah and Mäntylä, 1995)

In addition, the persistent naming problem is also a topic of research effort. When a topological entity is referred to by an operation, a unique identifier is attached to the referred topological entity for retrieving it later. However, in the re-evaluation of the feature model, the referred topological entity may be modified or deleted, so the

identifier cannot be used to retrieve the correct topological entity, which has been a problem in feature-based design for years. In the subsequent two subsections, the persistent naming problem and the history-based modeling mechanism are reviewed in details.

2.1.2.2 Naming and Matching of Topological Entities

During the design process, the boundary entities of the intermediate B-rep model such as faces, edges, and vertices are usually referred to by the new design operations for the following purposes:

- As the operational object of a feature, i.e., the topological edge of a chamfer operation.
- As the attached object of a feature, i.e., the datum plane of the sketch of a sweeping feature.
- As the dimensional object of a feature, i.e., the positioning edge of a feature.

However, the referred topological entities may be modified during later modeling operations due to the interacting relationships between features. This phenomenon will result in some problems, e.g., generating undesired shapes, loss of reference entity, etc. during the re-evaluation process, which is termed the persistent naming problem.

Many research studies have been reported in the naming and matching mechanism. A survey of the major solutions of the persistent naming problem has been reported by Marcheix and Pierra (2002). The boundary entities of each feature can be named unambiguously using the feature's generating mode, and the interacting entities need to be discriminated by some topological and geometric information. In the work reported by Capoyleas *et al.* (1996), the boundary entities were named by the feature's

generating mode, and the ambiguities were removed by the topological context and the orientations of the entities. The matching of the entities was realized through a local comparison of the respective topological neighborhoods (Chan and Hoffmann, 1995). In the work reported by Wu *et al*. (2001), the boundary faces of the feature shape were named according to the feature's generating mode and their locations in the feature. The ambiguities of the interacting entities are removed by their parametric values on the adjacent faces. The limitation of this naming algorithm is that the arrangement of subdivided faces seems to be very sensitive to geometric and topological variations. A semantic naming scheme was reported by Wang and Nnaji (2005), where all the topological entities were named using the construct relations of the feature shape surfaces. All the surfaces are named and recorded persistently by a naming server, and the gradient information of the intersection curves is used to remove the ambiguities caused by non-linear surfaces. This approach provides an effective way to name and match the topological entities, since the gradient information can discriminate all the interacting entities clearly. For the matching approaches, the reported works can be classified into local matching method and global matching method. In the global approach, the matching is carried out by the comparison and mapping of two sets of entities, which are the entities resulting from the initial model and the entities from the re-evaluated model. In the local approach, only the entities referred to in the initial model is compared with the set of entities resulting from the re-evaluated model.

2.1.2.3 Boundary Evaluation in Feature-based Design

Boundary evaluation is a key process in feature-based design, and its working principle has been well addressed in literature (Keyser *et al*., 2004; Requicha and Voelcker, 1985). The evaluation process consists of two working stages: at the first stage, the

boundary faces of the B-rep models are intersected pairwisely, partitioning them into separate sub-faces according to the intersection curves; at the second stage, the partitioned faces are identified and selectively stitched to the resulting B-rep model. As shown in Fig. 2.5, the cylindrical shape of a *ThroughHole* is subtracted from the *Block*, where the top face f_3 is attached to face f_1 and the bottom face f_5 is attached to face f_2. At the first stage, the intersecting faces $f_1 \cap^* f_3$ and $f_2 \cap^* f_5$ are computed to generate the partitioned faces $f_{1.1}$, $f_{2.1}$. At the second stage, the top face f_1 and the bottom face f_2 are replaced by $f_{1.1}$ and $f_{2.1}$ respectively, and the new face f_4 of the *ThroughHole* is stitched to the new resulting B-rep model.

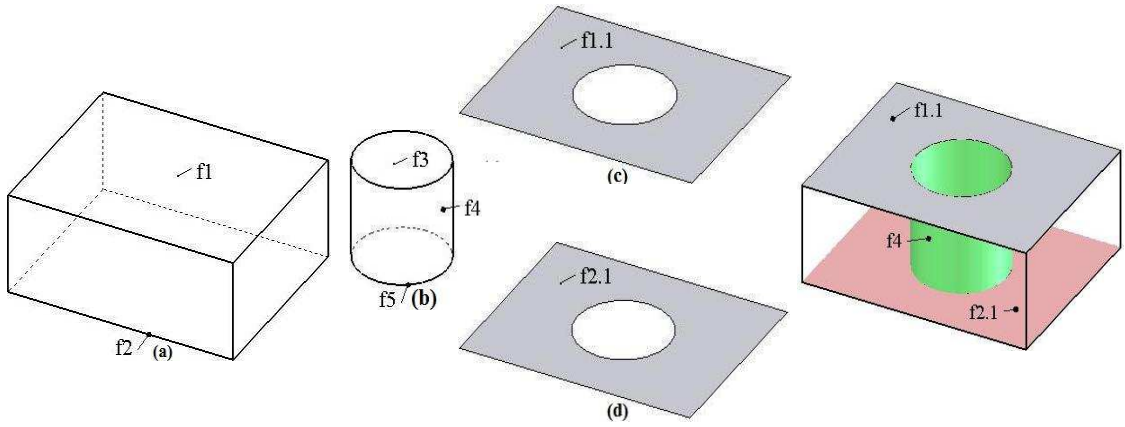


Fig. 2.5 Boundary evaluation process

In order to save the computation cost in the boundary evaluation in feature-based design, two methods have been devised and developed, namely, storing all the intermediate B-rep models at each history step, and storing only the deltas between the history steps (Bidarra *et al*., 2005). If the intermediate B-rep models at each step are stored, it requires a large amount of storage space. As shown in Fig. 2.6, when a feature is modified, e.g., the feature at step5, the modeling evaluator will go back to step5 and re-execute the model history based on the intermediate B-rep model stored at step4. In this case, all the intermediate B-rep models at each step would need to be

stored during the design. If the deltas are stored, it typically requires less storage space, but the computation cost for rolling back from the current B-rep model to the stage based on which the model history is re-executed is high. As shown in Fig. 2.6, the delta at each history step is smaller than the corresponding intermediate B-rep model. The former approach is currently being used in most of the feature-based design systems, in which only the history steps later than the edited feature node need to be re-executed after each modification. As shown in Fig. 2.6, since the intermediate B-rep model at step4 is stored, only the operations from step5 to step10 are re-executed when the feature *Pad* is modified. However, as observed from the DAG of the design features in Fig. 2.4, the features created later than *Pad* are irrelevant to the *Pad* modification. Consequently, the improved modeling approach is still not a desirable solution for the re-evaluation of the model. The computation complexity of this improved approach has been analyzed and reported by Bidarra *et al*. (2005), where the computation cost was analyzed using three representative models for the 'add feature', 'remove feature' and 'modify feature' operations. For the 'add feature' operation, the computation time includes three aspects, namely, identifying the intersecting boundary faces, Boolean operation of the intersecting faces, and updating the resulting B-rep model. For the 'remove feature' operation and the 'modify feature' operation, the main computation time is the time associated with re-adding the feature shapes that are created later than the feature being removed or modified.

The problems caused by the static chronological 'feature creation order' was solved using a cellular representation and modeling scheme reported by Bidarra and Bronsvoort (2000), where the non-associative set operations (union and difference) were replaced by a non-regular union operation. The proposed union operation makes

the ‘feature creation order’ irrelevant to the resulting cellular model. Thus, the computation cost for the ‘remove feature’ operation and the ‘modify feature’ operation is solely dependent on the number of the features being modified and their overlapping features. However, due to the complexity of the cellular model required for a complex model, this approach is not scalable in practice (Hoffman and Joan-Arinyo, 1998a).

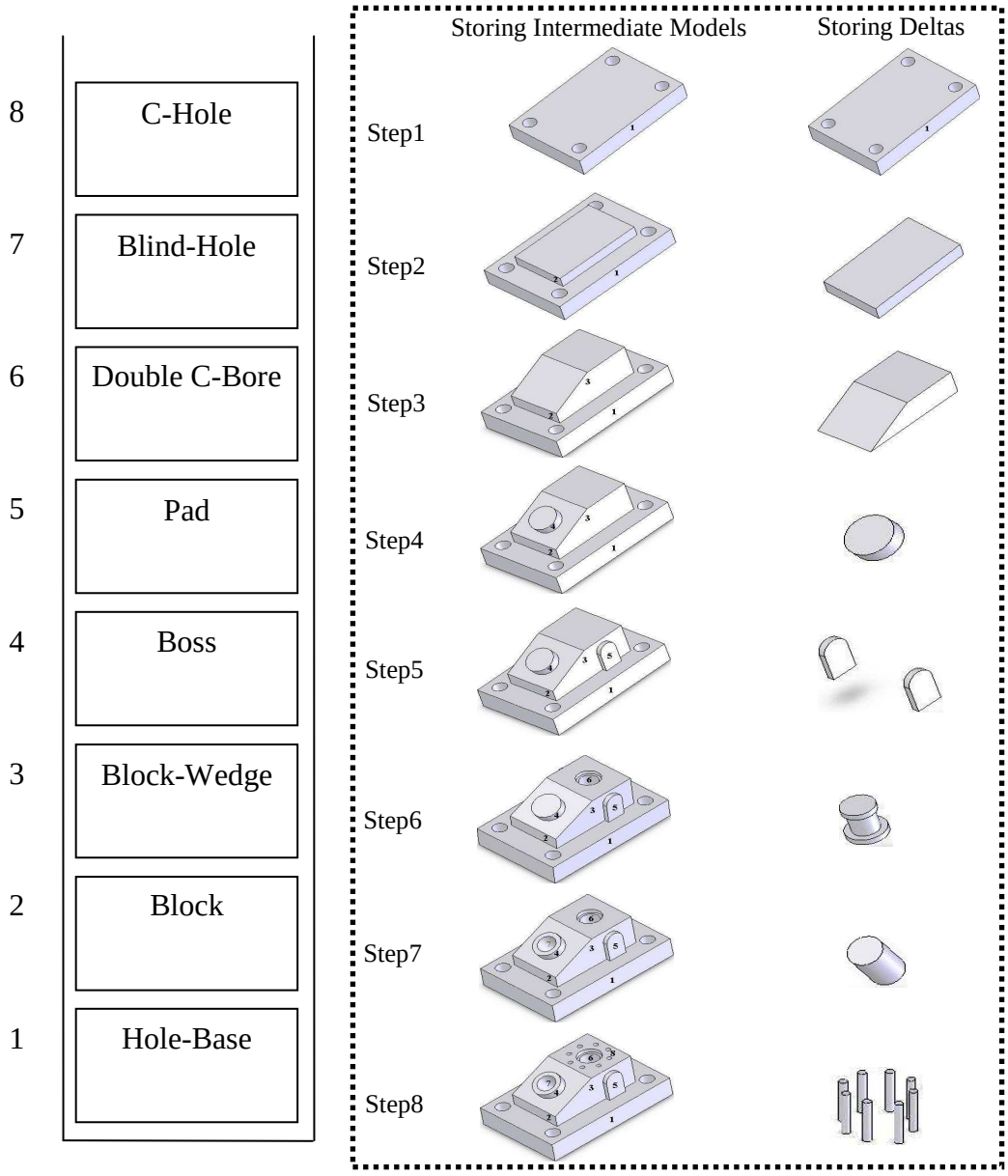


Fig. 2.6 Two improved modeling approaches

Based on the above reviews in section 2.1.2, it can be seen that current feature-based design has several weaknesses and shortcomings, e.g., persistent naming problem, high

computation cost, etc. The main shortcomings are caused by the history-based modeling procedure, which needs much research effort. If the 'feature creation operation' in the model history can be adjusted dynamically during the design, many of the above problems would be solved. As such, in this work, the boundary evaluation mechanism is investigated and addressed.

2.1.3 Freeform Feature Modeling

The success of a new product depends not only on high quality and short development time, but also on its attractive and pleasing appearance. Hence, freeform surface modeling is popularly used in aesthetic and engineering product design, in which the freeform surfaces are described using Bézier, B-spline and Non-Uniform Rational B-Spline (NURBS) curves and surfaces (Piegl and Tiller, 1997). Current Feature modeling can be adapted into freeform surface modeling for facilitating users to manipulate freeform surfaces intuitively, which is termed freeform feature modeling. In this section, freeform feature modeling is first introduced. Secondly, the definition and specification of freeform features are reviewed. Thirdly, the applications of displacement features in product design are investigated and discussed.

2.1.3.1 Introduction of Freeform Feature Modeling

The definition and modification of freeform surfaces require a deep knowledge and great skill in the manipulation of the underlying mathematical models (van den Berg *et al*., 2002), e.g., the control points, knot vectors, etc. As a result, many high-level manipulation tools and methods have been proposed, e.g., a mechanical-based deformation technique (Leon and Trompette, 1995; Pernot *et al*., 2005), a dynamic NURBS (Qin and Terzopoulos, 1996), a surface representation model (Zhang *et al*., 2004), a deformable freeform feature template (Song *et al*., 2004), and a feature shape

transposition approach (Langerak, 2008). Concurrently, some researchers have attempted to adapt the feature concepts in freeform surface modeling (Pernot *et al*., 2008; van den Berg *et al*., 2002), termed freeform feature modeling. This freeform feature modeling technique defines generic freeform shapes in combination with intuitive and user-friendly parameters, e.g., performing standard modeling operations and setting high-level constraints on certain geometric elements (3D points, curves) (Nyirenda and Bronsvoort, 2008, 2009; van den Berg *et al*., 2003), connecting certain Bézier surface patches and associate them with high-level parameters (Vosniakos, 1999), specifying the set of all possible parametric configuration of a shape configuration (Langerak, 2008, 2009); thus the freeform surfaces can be created and modified by specifying certain intuitive parameters.

In freeform surface modeling, usually a structural surface is given, and some operations are performed on the base surface, e.g., adding surface patches, and deforming or removing the surface regions. Hence, the freeform surface features are related to these operations, and some classification schemes from this point of view have been reported. In the work reported by Fontana *et al*. (1999), the freeform features in aesthetic design are classified into two categories according to the different phases of the design activity, namely, structural features and detail features. The structural feature is used for defining the surfaces constituting the product, and the detail feature is used for modifying the local regions of the structural surfaces, including deformation features and elimination features. The deformation features can be further classified according to the topological and morphological properties of the deformed regions, including border, channel, internal, extrusion, and intrusion features. The elimination features are classified according to the smoothness and topological

properties of the removed regions, including sharp, finished cuts, inlet, hole, and gap features. A similar taxonomy was reported by Nyirenda and Bronsvoot (2005), where the freeform features are grouped according to the geometric characteristics, including deform, cut, and transition features, and are grouped according to the topology of features, including border, channel and internal features. In the work reported by Sunil and Pande (2008), the features on a freeform sheet metal part are identified by studying the commonly used operations in sheet metal parts production. The freeform features are classified into face-based, edge-based, and transitive features. Face-based features lie on a face, edge-based features lie on the periphery of the part, and transitive features lie between faces. The above categories indicate that freeform features are related to the operations that modify the local regions of a base surface, and freeform feature modeling is basically to encapsulate the relevant operations in a high-level user interface.

2.1.3.2 Specification of Freeform Features

From the geometry point of view, freeform features can be classified into freeform surface features and volumetric freeform features. As the specification of a regular feature, a freeform feature would comprise of a generic shape description, the parameterization of the shape, and the validity conditions. The shape can be described as a construction procedure (procedural approach), or described as some geometric constraints on certain geometric entities (declarative approach). The difference from regular features is that the boundary of the freeform features is described in terms of freeform curves and surfaces. The parameterization of a freeform shape is not as simple as a regular shape, in which certain user-friendly parameters should be mapped to the underlying geometric representation directly.

In the work reported by Vosniakos (1999), the boundary of a freeform feature is determined by examining the various components within a product family. Each freeform surface feature is composed of several 4×4 Bezier surface patches that are connected through some geometric constraints. The high-level parameters are assigned to the feature shape, which perform directly on the control points of the constituent surface patches. In this approach, when a constituent surface patch is modified, other surface patches most probably would need to be modified as well, and this poses a big drawback. In the work reported by Nyirenda (2006), the generic shape of a freeform surface feature is defined by some Freeform Feature Definition Points (FFDPs) that are points in 3D space. The geometric curve can be determined easily by interpolating a set of FFDP. Analogously, the shape of a freeform surface feature can be determined by interpolating the geometric curves using the standard interpolation algorithms, e.g., lofting, skinning, etc. High-level parameters are assigned to the locations of the key FFDPs, thus all the remaining FFDPs can be positioned by input parameters and deductive parameters. This approach fails to address the question of combining the freeform features together to form a part model.

A feature specification approach for volumetric freeform shapes was reported by van den Berg *et al*. (2003). The profile and trajectory of a sweeping shape are both geometric curves that are determined by interpolating certain FFDPs, from which a volumetric freeform shape is generated by using the standard sweeping operation. For positioning the FFDPs, a network of geometric constraints, including distance constraints and angle constraints, are defined and solved. The geometric constraints can be related by algebraic constraints for defining certain high-level parameters. In volumetric freeform features, since the boundary surfaces are non-planar, feature

attachment operations are not as straightforward as that in regular-shaped feature attachments. In the recent work reported by van den Berg and Bronsvoort (2007), an attachment approach for freeform extrusion features was presented. In this approach, the initial feature shape that is not seamlessly connected to the attach face is extended, thus the extended feature shape can intersect with the attach face completely. In another work reported by van den Berg *et al*. (2004), the freeform shape is created by wrapping certain cross-sections defined by 3D points. Since the cross-sections have enough degrees of freeform, the general cross-section can be deformed to intersecting with the target surface. However, this approach is not plausible since the cross-section generated by interpolating 3D points does not lie on the attach surface seamlessly.

From the above studies, it indicates that the definition procedure of a freeform feature is similar to that of a regular feature: firstly create a generic shape; secondly associate certain intuitive parameters and constraints with the generic shape. However, attachment operations here are quite complicated since the boundary surface is not planar any more. In addition, specifying freeform features based on certain 3D points violates the essence of freeform modeling, since the freeform shape cannot be modified flexibly in this case. In this work, the specification and taxonomy of freeform features are not the key points. The focus here is the displacement feature modeling, which is reviewed in the following subsection.

2.1.3.3 Displacement Features in Product Design

Displacement feature is a type of freeform surface feature that deforms a region of the base surface. Embedding a number of displacement features into a base surface is quite common in industrial parts, e.g., automobile inner panel, airplane, refrigerator, etc.

(Cavendish and Marin, 1992, 1995). For displacement features, a modified region of a given surface is displaced towards the exterior or interior of this surface, after which it is blended with the unmodified surface region (Nyirenda *et al*., 2005). The modeling procedure generally includes three steps. Firstly, the modified region is defined by setting a boundary curve on the base surface. Secondly, the surface region inside the boundary curve is trimmed and displaced towards the exterior or interior of the base surface. Lastly, a blending surface is generated for connecting the displaced surface region and the un-modified region, as illustrated in Fig. 2.7.

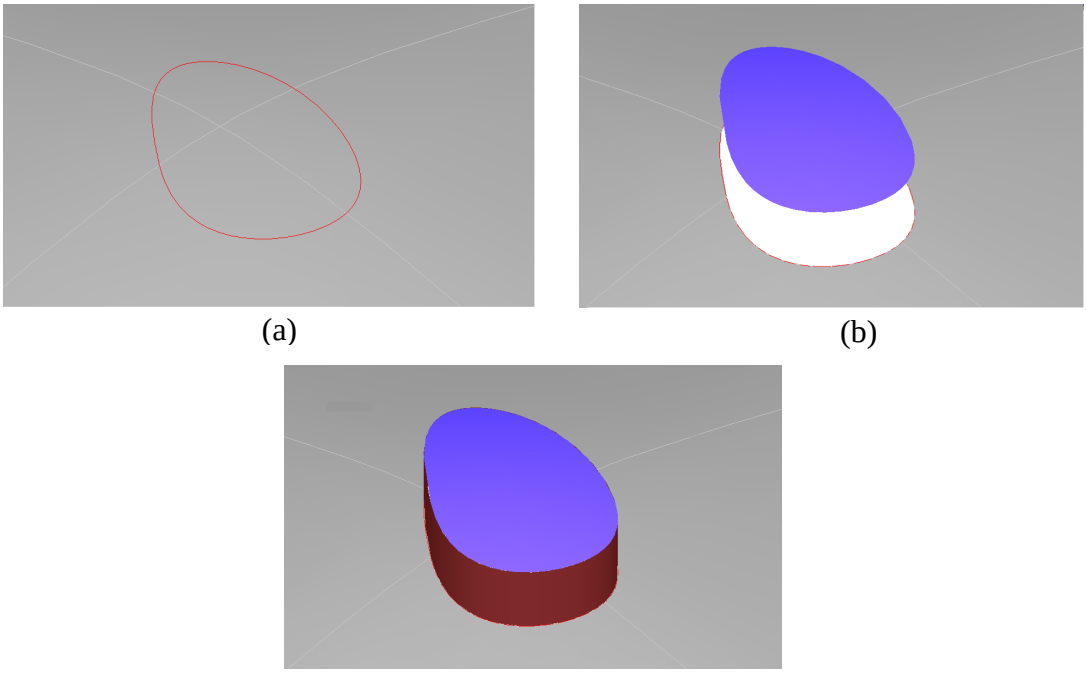


Fig. 2.7 Displacement feature modeling: (a) boundary curve; (b) displaced modified region; (c) blending surface

Some studies have been conducted in this modeling procedure. In the work reported by Cavendish and Marin (1992), the boundary curve on the base surface and the boundary curve of the modified surface region are designed in the plan view drawing, and the blending formula is the interpolation of the implicitly given surfaces. In the work reported by van Elsas and Vergeest (1998), the boundary curve on the base surface is

sketched by the designer and the transition surface is generated using the Cubic Hermite Interpolant, in which the tangential continuity (G^1) is approximated.

For a displacement feature, the boundary curve is basically a 3D curve lying on the base surface. In this circumstance, the 3D curve is represented explicitly and the representation is control-point based. In general, the 3D curve is first represented as a curve in the parametric domain of the base surface. Next, it is evaluated in the base surface as a space curve. The exact curve on a freeform surface in the control-point based representation can be computed using several approaches, e.g., point sampling and interpolation, power basis conversion, direct Taylor expansion, and polar forms (Renner and Weiβ, 2004). However, the degree of the exact curve is high, which could result in computationally demanding evaluation and may introduce numerical instability. Approximations are used to overcome this problem, where a lower degree curve is approximated within the user-specified tolerance (Renner and Weiβ, 2004; Yang *et al*., 2004). In this work, the boundary curve on the base surface is an exact curve rather than an approximated curve, which ensures that the continuity across the boundary is at least positional continuity (G^0).

Surface blending is used for replacing sharp edges with smooth surfaces, or creating smooth surfaces between a pair of non-intersecting surfaces. A survey on parametric blending methods has been reported by Vida *et al*. (1994). Whited and Rossignac (2009) recently reported a brief survey on the blending methods, and proposed a set-theoretic formulation for variable-radius blending, in which a "bounding" solid is used to control the radius of the rolling ball locally. In order to achieve the tangent plane continuity, the Cubic Hermite Interpolant has been adopted for surface blending (Elber, 1997; Elber, 2005; Kim and Sprynski *et al*., 2008; van Elsas and Vergeest, 1998). In

the Cubic Hermite Interpolant, the interpolating surface is basically a $n\times 3$ Bézier surface patch, where n is the degree of the boundary curve. The critical issue is the selection of the tangent curves that guarantee the tangential continuity across the boundary curves. Five methods were introduced by Kim and Elber (1997) for determining the tangent curves symbolically. However, the polynomial degree of the tangent curve can be as high as $(2nm-1)^3$, where m is the degree of the base surface in u and v directions. In order to obtain a surface with a low degree, the degree of the tangent curve should be reduced. Since the reduction in the degree of the tangent curve is a non-trivial task, determining the tangent curves directly is not a desirable solution. To avoid determining the tangent curve symbolically, van Elsas and Vergeest (1998) proposed an approximation method, where a set of points are sampled on the parameter curve uniformly, and the tangent vectors at the sample points are used to position the two interior rows of control points in the blending surface. However, this approximation method does not ensure that the tangent vector of the t isocurve in the blending surface would lie on the tangent plane of the base surface, which means that the tangent plane continuity cannot be achieved. In this work, the tangent curve is obtained by interpolating the sample points that are on the respective tangent planes of the base surface. Hence, this ensures that the blending surface patch contacts the base surface tangentially.

From the above reviews in section 2.1.3, it can be seen that current freeform feature specification is not reasonable to some extent. Freeform features should be created and manipulated intuitively by the users. More importantly, the modeling flexibility should not be restricted. In this work, the displacement feature modeling is studied, in which

the generation of the boundary curve and the surface blending approach would need more research effort.

2.2 Collaborative Computer-aided Design

Product design and manufacture has been shifted to a collaborative activity, where a group of designers from several departments or companies work together to develop a complex product. In this situation, a collaborative framework is strongly needed for integrating and coordinating the designers from different domains. Much research has been actively conducted in this field to develop new approaches and systems supporting collaborative design activities. The collaboration was categorized into three types by Li and Qiu (2006), namely, visualization-based collaboration for conceptual design and product review, cooperative creation and manipulation (co-design) for detailed design, and concurrent engineering integrating the design and the related manufacturing processes.

In visualization-based collaboration, all modeling functions and native 3D models, e.g. B-rep models, reside in the server. This collaborative mechanism only supports visualization, annotation and inspection of the product model at the macro-view. It is suitable for the on-line team to take on design discussion, product review, design remarks and conceptual design. The transmitted product model in such a collaborative environment is usually represented as a meshed model, which is small-sized over a B-rep model and can be used for visualization and some analysis applications directly. However, normally, the meshed model does not contain all the product information and design intent, so the designers cannot interrogate and manipulate it as a native 3D model.

The other two collaboration systems are presented in the following sections. Firstly, concurrent engineering which is also termed Computer Supported Collaborative Design (Shen *et al*., 2008) is investigated. Secondly, the studies in the co-design environment are reviewed, including the coordination of design operations and product information synchronization.

2.2.1 Computer Supported Collaborative Design

Computer Supported Collaborative Design (CSCD) is one of the concepts to re-organize the design process with objectives for better product quality, shorter lead-time, more competitive costing and higher customer satisfaction (Shen *et al*., 2008). In CSCD, the multidisciplinary design teams, including conceptual design, detailed design, manufacture, testing, simulation, etc., are integrated and coordinated in product development. Hence, the design conflicts can be identified in the early phase of product life-cycle, and the lead-time can be shortened. Since the design teams in CSCD may be geographically distributed in an enterprise or across several enterprises, the application modules used in the development process need to be integrated as a distributed and collaborative system using the new IT and communication approaches, e.g., Agent Technology, Web Services, etc.

In CSCD, the integration between computer-supported applications requires product information exchange within the integrated environment. Two solutions have been proposed to provide a compatible product model, namely, a neutral file-based model or a central master model involving all relevant product data.

International Standard for the Exchange of Product Data (STEP) offers the desired neutral specifications for product models with its open and extensible structures. STEP data files can be read directly by STEP processors, applications or through STEP access interfaces. As the distributed system architecture proposed by Zhou and Nagi (2002), STEP was used for information modeling and mapping for a virtual enterprise. However, STEP has some drawbacks that hinder its applications in current distributed systems. Firstly, the interface between applications is static thus the entire product model needs to be re-transmitted once some changes are made on the original model. Secondly, it is still shape-based centric that provides insufficient product information, where design intent, namely, parameters, features, and constraints, cannot be exchanged based on current STEP. In order to exchange parametric models, some approaches and standards have been proposed, such as Enabling next generation, Part 108, Part 55, and solid model construction history, etc. (Mun *et al*., 2003).

For the integrated system using a central product model, all the relevant product information and data processing are deposited in a central repository so that specific applications can access and manipulate the specific data subset. Hoffman and Joan-Arinyo (1998b) proposed a product master model for coordinating the CAD system with the downstream application processes. The master model is a repository that maintains the integrity and consistency of the deposited information on the geometry data. In the master model, the CAD system deposits the net shape, and other domain-specific applications can retrieve the shape elements and can deposit processed information on the net shape. When a specific shape element is changed by a client, a change protocol is used to inform the other clients of the shape change. The mechanism for maintaining the consistency between the distributed product views was elaborated and extended in their later work (Hoffman and Joan-Arinyo, 2000). Martino

et al. (1998) reported an integrated feature-based modeling approach for the integration between the design process and the downstream engineering processes, which provides a homogeneous, multiple view feature-based representation of the product model. A certain application can extract specific feature-based model from the central model, and request for modifications. In order to avoid conflicts, the designer is the only user allowed to modify the product model and propagate the model change.

In addition to a natural product model, the essence of the distributed and collaborative system is that the engineering tools are encapsulated as web-enabled services, thus they can be delivered, discovered, integrated, and interoperated dynamically. Three main technologies have been developed to distribute and integrate the manufacturing resources, namely, Web technology, Agent technology and Web Services. In Web-based collaborative design, engineering tools are encapsulated as Web-enabled modules, thus the designers at the client sides can share information and invoke the engineering tools across the Internet. In Agent-based collaborative design, engineering tools are encapsulated as agents that have the capabilities of being autonomous, coordinative, communicative, intelligent, etc., thus the specific agent-based engineering tools can be integrated as a collaborative environment. Similarly, in Web Services systems, engineering tools are encapsulated as Web Services, thus they can be delivered, discovered, and integrated dynamically through the ubiquitous Internet system. In addition, the combination of Agent Technology and Web Services can also be used for developing collaborative engineering systems.

Many studies have been reported in the exploration of integrated systems using the above technologies. A design service marketplace was developed by Abrahamson *et al.*

(2000), in which organizations can publish, subscribe, and manage the solution services. A component-based framework for advanced CAD/CAM applications using the component technology was developed by Liu (2000). The interface component encapsulates the feature data and provides a set of interface functions for the access and manipulation of the internal data, which decouples the developments of specific applications. An open system was developed by Gerhard *et al*. (2001) to provide solutions for rapid integration of design and manufacturing modules. The explicit interface and explicit access methods of the Event-based mechanism guarantee the de-coupling of application development and implementation. Li *et al.* (2004b) reported an Internet-enabled collaborative and concurrent engineering design system based on Java Servelet, integrating three functional modules, namely, co-design, web-based visualization, and manufacturing analysis. In this system, an event-based mechanism is proposed to maintain asynchronous communication among the three modules. An agent-based collaborative design environment was developed by Hao *et al*. (2006). The actual engineering software tools are encapsulated as problem-solving agents (PA). A design work is defined as a job agent, which contains the workflow of the requested PAs. When a job agent is executed, the involved PAs interact and communicate automatically using the XML based message. In the work reported by Kuk *et al*. (2008), each of the engineering software is wrapped and offered as a service via Web Services, and is consumed and invoked by a process/analysis agent. Web Services are used for integrating world-wide distributed resources, and agents are used for the cooperation and coordination mechanisms for the engineering activities.

From the above reviews, it can be seen that computer-supported collaborative design has been a topic of research effort in the past few years. The integration of different

applications is realized using the new middleware modules and intelligent information technologies. The product information transferred across the integrated environment is represented as a neutral file-based model, or the information is stored and managed in a central master model. However, since the application services need to be integrated dynamically, current middleware technologies sill need much research effort. In addition, STEP needs to be extended to include more high-level product information.

2.2.2 Collaborative Feature Modeling

The co-design system is usually termed collaborative feature modeling, where a group of designers manipulate the product model concurrently. Two types of architecture for co-modeling system are usually adopted:

- communication server + modeling client
- modeling server + manipulation client

In the first architecture, each designer is provided with the whole modeling capabilities and a communication interface. The server coordinates the design session through receiving and broadcasting action events. In the work reported by Chan and Ng (2002), each client holds the whole modeling functions and a copy of the central model. Once a primitive object is edited by one designer, the design event, consisting of design action and design object, is forwarded to the server, and is then broadcast to other co-designers by the server. Li *et al.* (2007) proposed a mechanism to integrate heterogeneous CAD systems. An add-on for the translation between specific modeling operations and neutral modeling commands is embedded into specific CAD systems. Hence, the modeling operations performed by one designer can be broadcast to other designers via the server. The limitation of the first architecture is that the user module contains all the modeling functions, so it is not flexible to be distributed across the

Internet. In the second architecture, most of the modeling work is performed at the server, whilst each user has limited capabilities to visualize and manipulate a simplified product model, for instance, the framework for web-based feature modeling (Bidarra *et al*., 2002), the client/server framework enabling a dispersed team to accomplish a feature-based design task collaboratively (Li *et al*., 2004a). The user module in this architecture can be distributed flexibly across the Internet. However, the designed model needs to be transmitted back and forth in this case, which raises many problems due to limited bandwidth.

In addition to the collaborative part design, co-modeling is also reported in assembly design, where each designer works on a specific part and the compatibility between different parts is maintained by the server. Shyamsundar and Gadh (2001) reported a geometric representation, termed AREP, for real-time collaborative assembly design. AREP consists of several assembly units (AUs), each of which comprises of interface assembly features and virtual design space (VDES). As such, each designer can focus on a specific VDES, and the compatibility is guaranteed by the Interface Assembly Features (IAF) in the VDES. The weakness of this approach is that VDES must be produced before the detailed design, which is difficulty for some products. Chen *et al*. (2004) reported an Internet-enabled real-time collaborative assembly modeling system, in which the product was represented as a client/supplier hierarchy. In the work reported by Kim *et al.* (2004), an assembly design (AsD) formalism and the associated AsD tools were developed, which can capture the joining relations and spatial relationships in assembly design. The AsD formalism specifies the assembly symbolically and the AsD engine generates the assembly model, so each designer only sends an AsD model rather the entire geometric model. The above three studies

indicate that the crucial issues in collaborative assembly design are dividing the product model into different parts and maintaining the compatibility between them.

In brief, co-modeling can be used in the detailed design of a part model or the assembly design. In this thesis, only the co-modeling of a part model is focused, where two issues should be addressed for employing it effectively. Firstly, a coordination mechanism is required for managing the concurrent design operations. Secondly, since the designers may be geographically dispersed in different locations, the modeling system becomes a distributed and collaborative environment. In this situation, the synchronization of the product model across the clients is a challenging issue.

2.2.2.1 Coordination Mechanism

Since an effective coordination mechanism is crucial for scheduling the collaborative design activity and resolving operation conflicts (Bidarra *et al*., 2002; Li and Qiu, 2006), exploring coordination mechanisms has been a research topic. In the literature, the locking mechanism, either total-locking mechanism or granular-locking mechanism, has been proposed to schedule the concurrent modelling operations, and some optimistic mechanisms used in group-editor systems have also been reported. In this subsection, the coordination methods reported in the previous works are discussed and the specific gaps that will be fulfilled in this research work are identified.

Some optimistic mechanisms, in which the object being edited is not locked, have been employed in the collaborative systems for coordinating the concurrent operations. The concurrent operations in the group-editor system can be classified into causality relation and compatible relation (Xue *et al.*, 2001; Imine 2008). The causality operations cannot be executed concurrently since they have to be executed in the same

order, but the compatible operations can be executed concurrently at different users' sites by using the operation transformation (OT) mechanism. Under these circumstances, users can perform the operations at the same time, after which the operations are classified and executed concurrently. Unfortunately, the classification mechanism is very complex in collaborative feature modelling as the modelling operations are difficult to be classified separately compared to that in group-editor systems. Jing *et al*. (2008) reported a no-locking mechanism for collaborative feature modelling. Each designer can perform a design operation locally, and then send the operations to the remote co-designers. A topological entity correspondence mechanism was proposed to resolve the operation conflicts. The reference entities of an operation are identified by a naming mechanism, and the lost operation entities of an operation are restored by rolling back and re-executing local operations. However, this no-locking mechanism is questionable in two aspects. Firstly, it assumes that there is no manipulation conflict, which means each designer edits different features at one time. This assumption does not hold in a distributed environment without any locking mechanisms. Secondly, it does not maintain the consistency of the feature creation orders at different design sites, and it assumes that the operations can be re-sorted and re-executed freely, which is not reasonable in current history-based feature modelling, as shown in Fig. 2.8. If the *Slot* creation precedes the *Boss* creation, the resulting model is Fig. 2.8(b), but the reverse execution order generates the model Fig. 2.8(c).

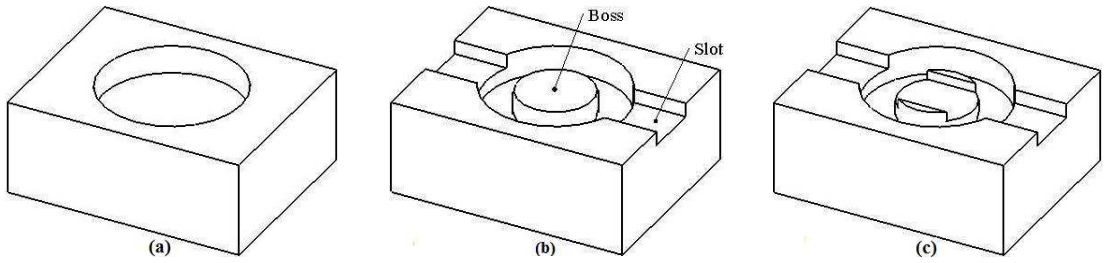


Fig. 2.8 Overlapping features

In view of the problems of optimistic mechanism, the locking mechanism is usually employed in collaborative feature modelling. By means of the total-locking mechanism, the entire product model is locked by the system and the control permission is dispensed to the designers in a sequential order by a coordinator. Bidarra *et al*. (2002) employed a 'traffic light' mechanism for coordinating simultaneous design operations. The design operations are queued at the client's side, and only one designer is permitted to submit his operation to the modelling server based on the status of his 'traffic light'. A control baton based mechanism has been adopted by some researchers for scheduling the collaborative design activity (Li *et al*., 2004; Li *et al*., 2007; Shen *et al*., 2006). At one time, only the designer who holds the control baton can edit the product model, while other designers only observe and receive the updated model information. The major drawback of the total-locking mechanism is that only one designer is permitted to edit the product model at any one time, thus the modelling is inefficient.

In order to overcome the shortcomings of the total-locking mechanism, granular-locking mechanism has been proposed and adopted in some reported works. In the work reported by Chan and Ng (2002), the shared product model was represented as a CSG tree, thus a node or a sub-tree of the CSG model can be taken as the locking granularity. This approach provides an important insight in that the product model can be divided into several independent portions. In this case, the designers do not need to obtain the full control of the working model; instead, they can work on different portions and then synchronize their modified portions. As such, several designers can edit and manipulate the product model at the same time, which provides an effective way for collaborative feature modelling. Li *et al*. (2008a, 2008b) employed a fine

granular-locking mechanism for feature models, where a feature model was divided into several scopes based on feature dependency relationships. The exclusive scope of a feature includes all its descendant features, all the ancestral features of the descendant features, and the feature itself. If the two features are not included in the exclusive scope of one another, they can be edited concurrently. However, this work has failed to address the issue of maintaining the exclusive feature creation order, and it did not consider the potential conflicts between design operations, as the problem shown in Fig. 2.9. Due to the position change of feature *cirSlot*, the reference edge e_1 of feature *Rib* diminishes and the *Rib* operation cannot proceed correctly.

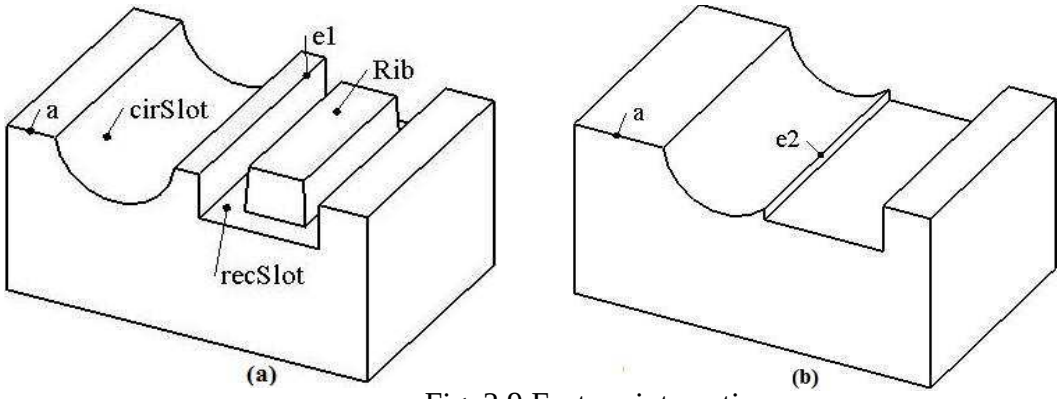


Fig. 2.9 Feature interaction

Based on the above review of optimistic mechanism and locking mechanism, it can be seen that the granular-locking mechanism is suitable for collaborative feature modelling. However, the reported works failed to address the following two issues: maintaining exclusive feature creation order and resolving operation conflicts. As a result, further research work on the granular-locking mechanism is imperative in collaborative feature modeling.

2.2.2.2 Product Information Synchronization

In the distributed and collaborative design environment, the critical problem is the dilemma between the large product model and the limited network bandwidth. This

problem also occurs in visualization-based collaborative design, where the meshed model is still very large to be transmitted over the Internet. In order to transmit the meshed model progressively, some simplification and refinement mechanisms have been proposed, e.g., 3D streaming. For streaming solid models, a cellular-based approach was developed by Lee *et al*. (2004) to generate progressive solid model (PSM). A feature-based solid model is represented as a PSM, consisting of a much coarser solid model together with a sequence of progressive features that are represented as a subset of feature cell faces. In each model sharing, the initial coarser model is transmitted first, and the progressive features are transmitted incrementally. However, in the progressive transition, the entire meshed model has to be transmitted repeatedly if changes are made on the original model. An innovative approach was proposed by Wu and Sarma (2004) to reduce transmitted mesh number rather than compressing it if changes are made on the meshed model. In this approach, the boundary representation space of a product model (B-rep K) is regarded as a finite set of cells, so changing a B-rep shape is equivalent to update a subset of cells of K. Based on this concept, the changed faces and meshed change model can be identified, and are further transmitted and merged with the old meshed model.

In the co-modeling system, the updated geometric model needs to be synchronized frequently across the co-designers. Transmitting the entire geometric model after each operation is infeasible in such a design context, so some research works have been carried out to transmit only the changed part of a model. Lee *et al.* (2001) developed a shape abstracting mechanism to provide each user an Attributed Abstracted B-rep (AAB) model that represents the central model on the server. The server transmits updated faces incrementally to the designers, and this reduces the network load

compared to transmitting the entire B-rep model. Li *et al.* (2004) proposed a distributed feature manipulation mechanism to reduce transmitted data size. Each time when a feature is edited, the server can filter the varied features and varied faces based on the feature interaction graph and broadcast varied information to the other designers. Through transmitting the varied faces instead of the entire CAD model, waiting time at the client side is shortened.

From the literature in collaborative computer-aided design, it shows that collaborative design has been commonly used in current product development, including visualization-based, co-modeling and concurrent engineering. In order to apply collaborative design effectively, much research work is required to address the issues involved in this system, e.g., dynamic integration of different application processes, coordination of design activities, and product sharing across the Internet. Specifically, for collaborative feature modeling, the two issues, namely concurrency control and model synchronization, remain as a topic of research effort.

In this chapter, the relevant studies in feature-based design and collaborative computer-aided design are surveyed and discussed. The literature review provides a substantial understanding of the problems in the relevant fields and the research issues addressed in this thesis.

Chapter 3 A History-Independent Modeling Approach

3.1 Introduction

The problems in current feature-based design have been investigated and discussed in the review section 2.1.2, they are due to the fact that the ‘feature creation order’ in the model history is static. In current feature-based design, the modification and evaluation of a feature in the model history depends strongly on the features created before the feature being edited, but does not depend on the features created later. This working principle contradicts with the operation performance from the user’s perspective, since the users always perform operations on the current B-rep model including all the features in the model history, and not the intermediate B-rep model which includes only the features created earlier. Hence, if the operation performance from the user’s perspective is consistent with the working principle of the modeling system, the problems in current feature-based design may be solved properly. A probable solution is as follows: firstly, single out the feature being edited and update the remaining features; secondly, re-attach the feature selected previously to the updated B-rep model. As such, the ‘feature creation order’ in the model history is changed, so the related problems can be resolved. The critical point is to update the remaining features such that the contribution to the B-rep model from the feature being edited is cleared. This update operation has the same effect with the ‘remove feature’ operation, but the working procedure is quite different. In the ‘remove feature’ operation, the B-rep model is updated by sequentially re-evaluating all the remaining features. In this devised procedure, only the intersecting features of the feature being edited are checked and re-evaluated, thus the computation time can be saved.

In this Chapter, the devised modeling operation, termed history-independent modeling approach, is elaborated and validated. Firstly, the working principle of creating a feature model is presented. Secondly, the feature intersecting relationship in a designed feature model is investigated. Thirdly, the working procedure of the proposed modeling approach is elaborated, including 'add feature' operation, 'remove feature' operation and 'modify feature' operation. Fourthly, the computational complexity of current feature-based modeling and that of the proposed modeling are analyzed, and the computation times are measured. Finally, the proposed approach is validated using a case study.

3.2 Feature-based Design

From the geometric perspective, feature-based modeling is a sequence of attachment operations of certain specific feature shapes, where the feature shape FS_i is combined with the intermediate part model PM_i through a regularized Boolean operation BO_i, as denoted in Eq. (3.1). The feature shape and the part model are usually represented as B-rep models, which consist of the topological entities and the underlying geometric entities. There are two types of Boolean operations, namely, union and difference operations. In the difference operation, the feature faces of the subtractive feature FS_i intersect with the boundary faces of the part model PM_i, and the intersected faces and the new feature faces are selectively stitched to the resulting part model PM_{i+1}.

$$FS_i < BO_i > PM_i = PM_{i+1} \tag{3.1}$$

As illustrated in Fig. 3.1(b), a *cirPocket* is subtracted from the initial *Stock*, the new feature faces f_2, f_3 and the intersected face $f_{1.1}$ are merged into the resulting part model. In the union operation, the feature faces of the additive feature FS_i intersect

with the boundary faces of the part model PM_i, and the intersected faces and the new feature faces are selectively stitched to the resulting part model PM_{i+1}. As illustrated in Fig. 3.1(c), a *Boss* is added on the bottom face of the *cirPocket*, the new feature faces f_4, f_5 and the intersected face $f_{3.1}$ are merged to the resulting part model. In a 'transition feature' operation, such as the chamfer and the fillet operations, the transition features can be converted into the additive or subtractive features. Thus, the execution process of the 'transition feature' operation is the same, where the new feature faces and the intersected faces are computed and selectively stitched to the resulting part model. As illustrated in Fig. 3.1(d), a *Chamfer* is basically a subtractive feature, which generates a chamfer face f_6 and modifies the incident faces.

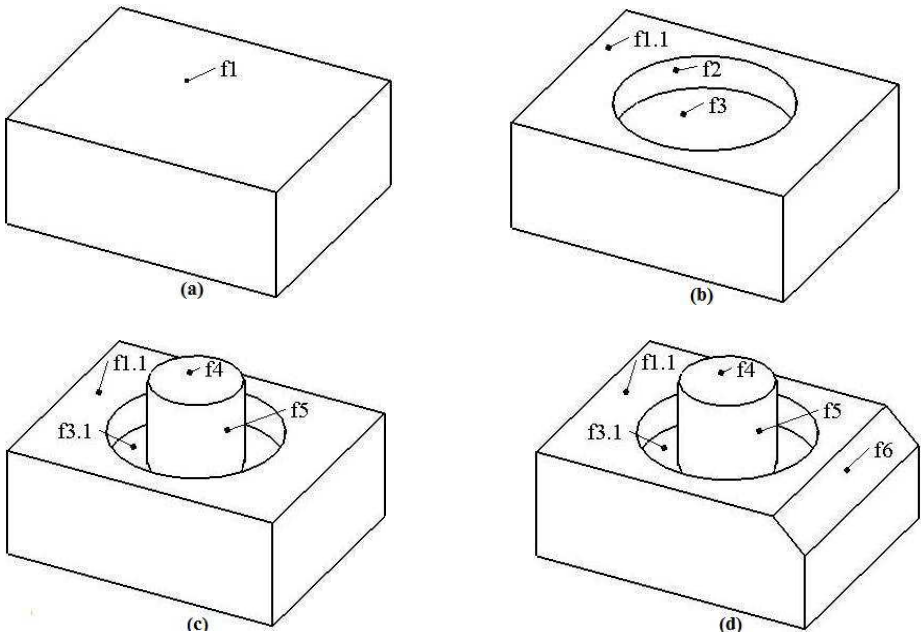


Fig. 3.1 Feature attaching process

It can be concluded that the feature attaching processes, which are the union, difference, and transition operations, are basically to update the boundary faces of the part model if the boundary representation approach is employed. In the resulting part model, all the boundary faces originate from the faces of the features, and there is no

boundary face that has no original feature face, which has been observed by Wu *et al.* (2001). As shown in Fig. 3.2, a feature *Boss* is first added to the top face of the initial *Stock*, and then a *cirPocket* is subtracted from the *Stock*. In SolidWorks, the resulting B-rep model is (b), where the face f of the model (c) is not on the model boundary since f has no original feature face.

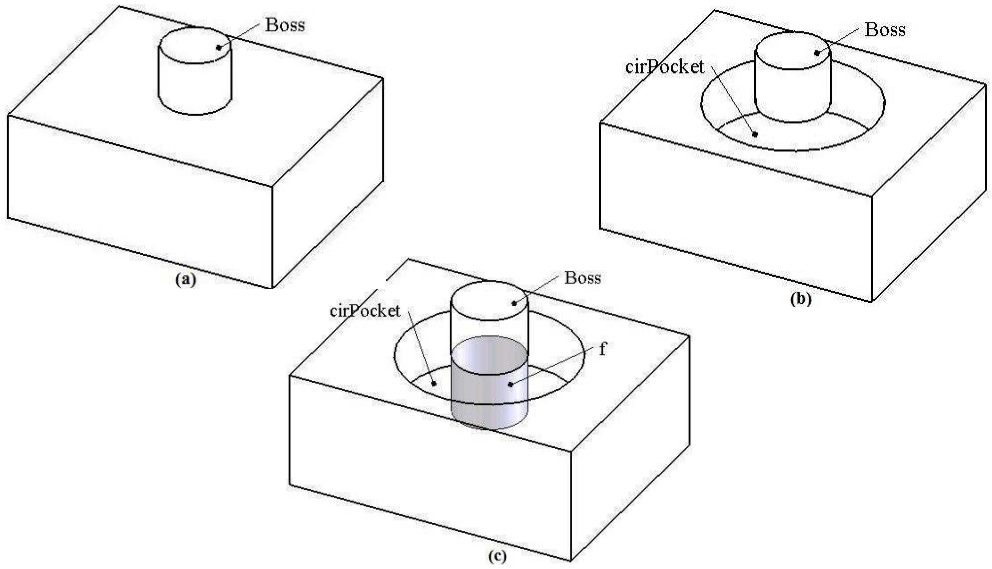


Fig. 3.2 'No original feature face' case: the Boss is floating on the model since f has no original feature face

3.3 Feature Intersecting Relationship

When a new feature F is created, the boundary faces of the intermediate B-rep model are modified due to the intersections with the faces of F, which are selectively trimmed and stitched to the resulting B-rep model. During the subsequent operations, the faces of F that are present on the model boundary may be further trimmed, split, merged or deleted due to the merging of the later features. As shown in Fig. 3.3, firstly an initial *Stock* is created, next a *rectSlot* is subtracted from the *Stock*, and lastly another *rectSlot* is subtracted. During the merging of the first *rectSlot*, feature faces $f_{(2.4,2)}$, $f_{(2.5,2)}$, $f_{(2.6,2)}$ are stitched to the model boundary, boundary face $f_{(1.1,1)}$ is split, $f_{(1.2,1)}$ and $f_{(1.3,1)}$ are trimmed. During the merging of the second *rectSlot*, feature

faces $f_{(3.2,3)}$, $f_{(3.3,3)}$ are stitched to the model boundary, f_2 and $f_{(2.4,2)}$ are split, $f_{(2.6,2)}$ and $f_{(3.6,3)}$ are merged, $f_{(1.4,1)}$ is trimmed.

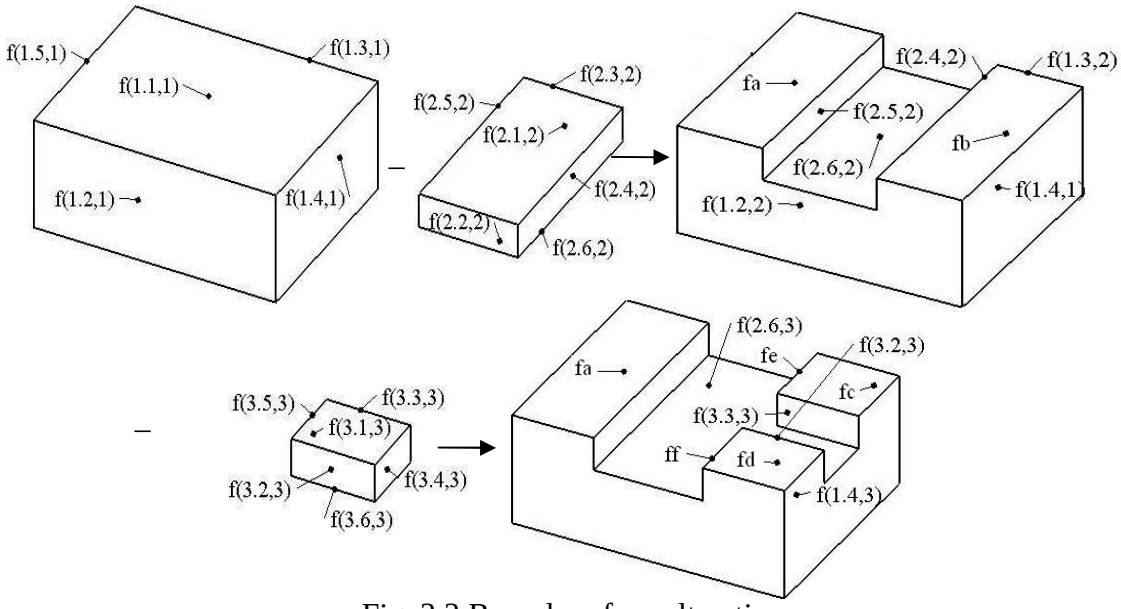


Fig. 3.3 Boundary face alteration

This alteration process can be illustrated with a *FaceIdGraph*, as shown in Fig. 3.4. Each face of a feature can be assigned a unique name in terms of the feature's generating mode and its location in the feature shape (Capoyleas 1996; Wu *et al*., 2001; Wang and Nnaji, 2005). Combining with the *FeatureId*, all the feature faces in the design model are named persistently, termed as the invariant name (*IN*). In the *FaceIdGraph*, each face is assigned a *FaceId*, in which the first item is the *IN* of the face and the second item is the *StepId* of this operation, as denoted in Eq. (3.2). In case that a face is split, the sub-faces can be discriminated in terms of the bounding feature faces and the geometric information of the intersection edges (Cripac, 1997; Wang and Nnaji, 2005), denoted in Eq. (3.3).

$$FaceId(f) = (IN, stepId) \tag{3.2}$$

$$FaceId(f) = (IN, INs(boundingFaces), stepId) \tag{3.3}$$

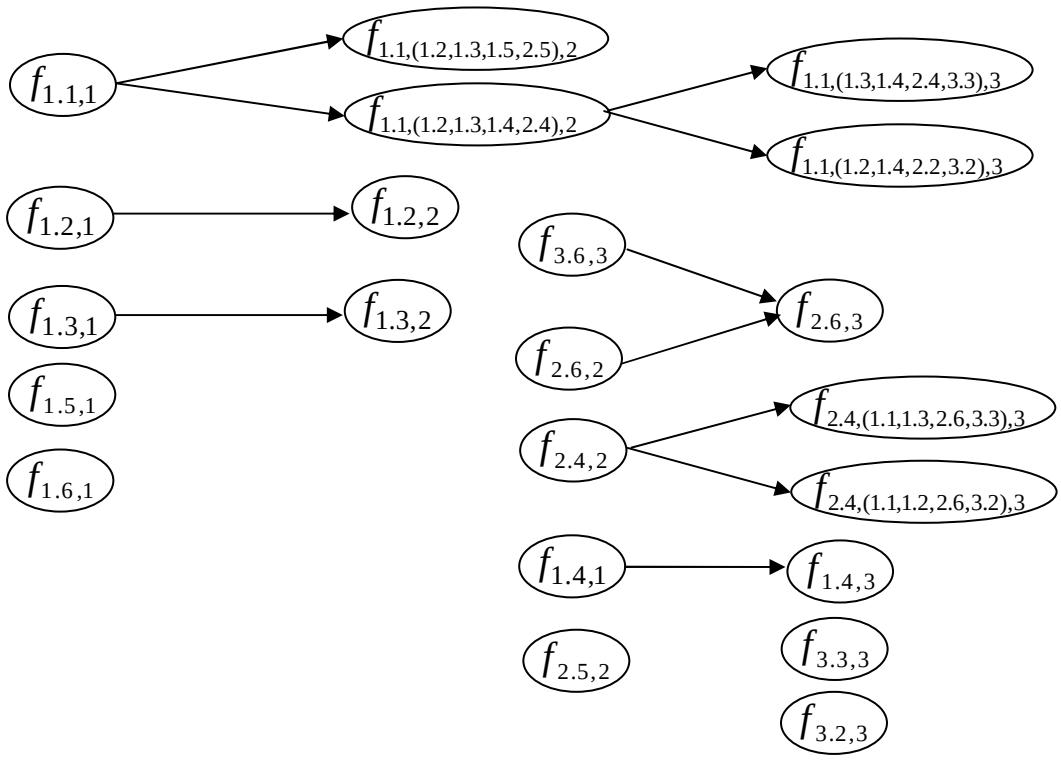


Fig. 3.4 Graph of altering faces

It can be seen that the faces of a feature that are present on the model boundary are not static, but are changed according to the model modifications. In this work, one can see faces originating from a feature present on the model boundary constituting the boundary contribution (*BC*) of this feature. All the *BCs* of the design features in a model constitute the boundary faces of the resulting B-rep model. During the design process, the *BC* of a feature is changed due to the intersecting relationships with other features, which are termed the intersecting features of this feature. As a result, modifying a feature in a product model is basically to alter its *BC* and the *BCs* of its intersecting features, so that the resulting B-rep model can be updated.

3.4 Proposed Feature Modeling Approach

3.4.1 'Add feature' Operation

When a new feature is attached to the product model, the feature constraints are first specified and solved, e.g., attach constraint, position constraint and dimension

constraint. Next, the feature shape is generated and combined with the current BRep model. In boundary evaluation, the intersecting faces in the current BRep model are identified and processed using Boolean operations first, after which the partitioned sub-faces and the new feature faces are selectively stitched to the resulting BRep model. In this research work, when a feature has been evaluated, there is an intersecting list recording to its intersecting features, and storing the intersection face portions between the intersecting faces and the feature faces, as shown in Eq. (3.4).

$$List(F) := [InterFeature(F); InterFacePortion(F)] \quad (3.4)$$

As shown in Fig. 3.5, the product model is developed by combining four features, F_2, F_3, F_4, F_5 to the initial feature F_1. During the development process, the intersecting features are recorded, and the intersection face portions are stored, as in Table 3.1.

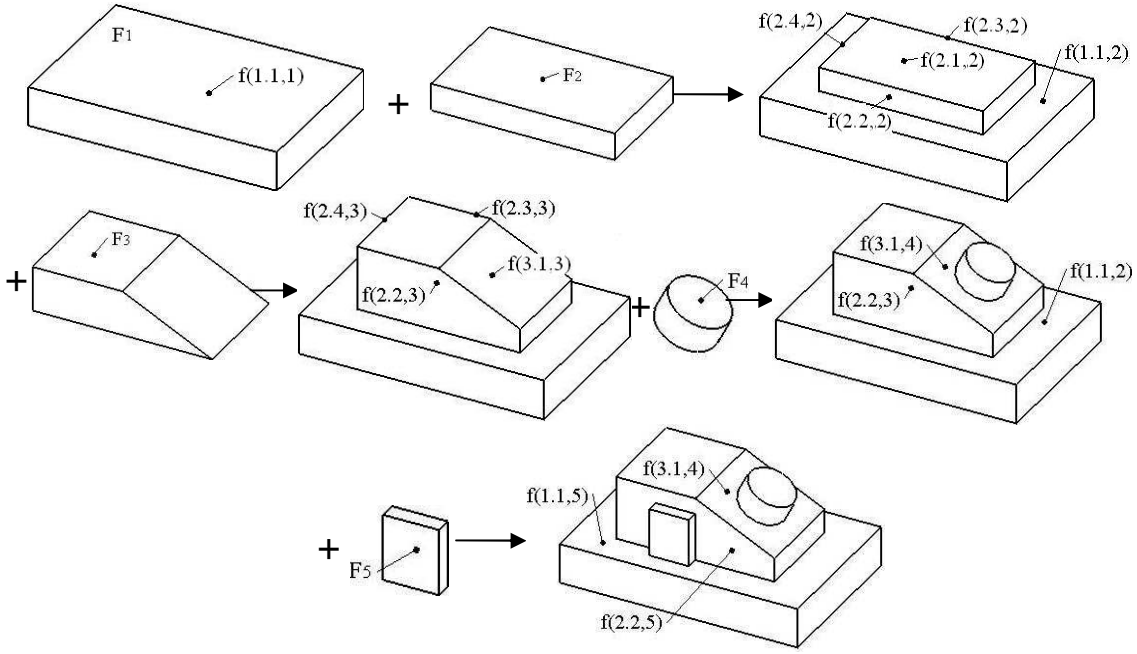


Fig. 3.5 'Add feature' operation#1

Table 3.1 Intersecting list #1

Intersecting List	F_1	F_2	F_3	F_4	F_5
Intersecting features	$F_2,\ F_5$	$F_1,\ F_3,\ F_5$	$F_2,\ F_4,\ F_5$	F_3	$F_1,\ F_2,\ F_3$
Intersection face portions		$f_{(1.1,1)} \cap^* F_2$	$f_{(2.1,2)} \cap^* F_3$	$f_{(3.1,3)} \cap^* F_4$	$f_{(1.1,2)} \cap^* F_5$, $f_{(2.2,3)} \cap^* F_5$

3.4.2 'Remove feature' Operation

In the 'remove feature' operation, when a feature F is removed, there are three major steps for updating the model boundary. Firstly, the boundary faces originating from F are removed from the model boundary. The boundary faces originating from F can be classified as follows: a) originating from F and being present on the model boundary as a topological face; b) being on the model boundary but is only a portion of a topological face, which is the merged face from faces originating from different features. The first type of faces can be removed directly, and the second type of faces should be updated by subtracting the face portions that belong to F. Next, the intersection face portions stored at the F creation step are merged to the model boundary, in which the impacts caused by the later intersecting features are considered. Lastly, the *BCs* of the intersecting features which are created later than F are updated by removing the impact from the removal of F. As shown in Algorithm #3.1, f_{FB} is the face defining the boundary of F; $f_{InterFacePortion}$ is the intersection face portions between intersecting faces and f_{FB} when F is being added; f_F is the present boundary face originating from both F; f_{merged} is the present boundary face that is merged from f_{FB} and the face belong to other features; $f_{InterFeatureB}$ is the face defining the intersecting feature F_{Inter}.

Algorithm #3.1:

$remove(F)\{$
  $update(BC\ of\ f_{FB})\{remove\ f_F;$
$update\ f_{merged}\ by\ (f_{merged} -^* f_{FB});\}$
  $update(BC\ of\ f_{InterFacePortion})\{$
    $if\ (!F_{Inter}): stitch\ f_{InterFacePortion};$
    $else: selectively\ stitch\ (f_{InterFacePortion} \cap^* f_{InterFeatureB});\}$
  $update(BC\ of\ F_{Inter})\{$
    $selectively\ GlueOrClear\ (f_{InterFeatureB} < BO > f_{FB});\}$
  $\}$

As shown in Fig. 3.6, feature F_5 in Fig. 3.5 has been removed from the current BRep model. Firstly, the boundary faces $f_{(5.1,5)}, f_{(5.2,5)}, f_{(5.3,5)}, f_{(5.4,5)}$ originating from F_5 are removed from the model boundary. Next, the intersection face portions f_a, f_b are merged to the resulting BRep model, which are the stored face portions $f_{(1.1,2)} \cap^* F_5$ and $f_{(2.2,3)} \cap^* F_5$.

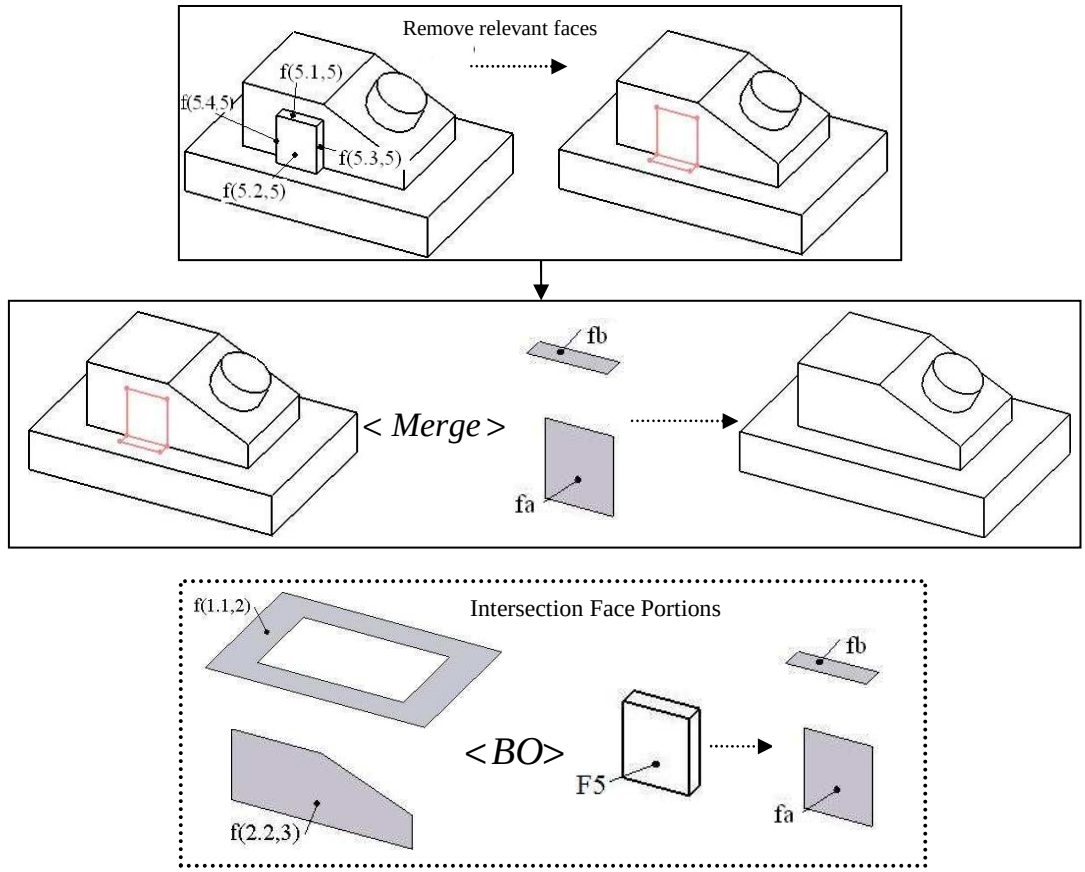


Fig. 3.6 'Remove feature' operation#1

When a feature F is removed, the features attached to F are usually removed as well. As shown in Fig. 3.7, when feature F_3 is removed, its attached feature F_4 needs to be removed as well. It is supposed the feature F_5 has not been combined, and the removal operations of F_3 are as follows. Firstly, the boundary faces $f_{(3.5,3)}, f_{(3.1,4)}$ originating from F_3 and the boundary faces $f_{(4.1,4)}, f_{(4.2,4)}$ originating from F_4 are removed from the model boundary directly. Then, the merged boundary faces $f_{(2.2,3)}, f_{(2.3,3)}, f_{(2.4,3)}$ are

processed using the Boolean operations with the feature faces defining F_3, and the merged boundary faces $f_{(2.2,3)}, f_{(2.3,3)}, f_{(2.4,3)}$ are replaced by the partitioned faces f_a, f_b, f_c. Next, the stored intersection face portion f_d are merged to the model boundary.

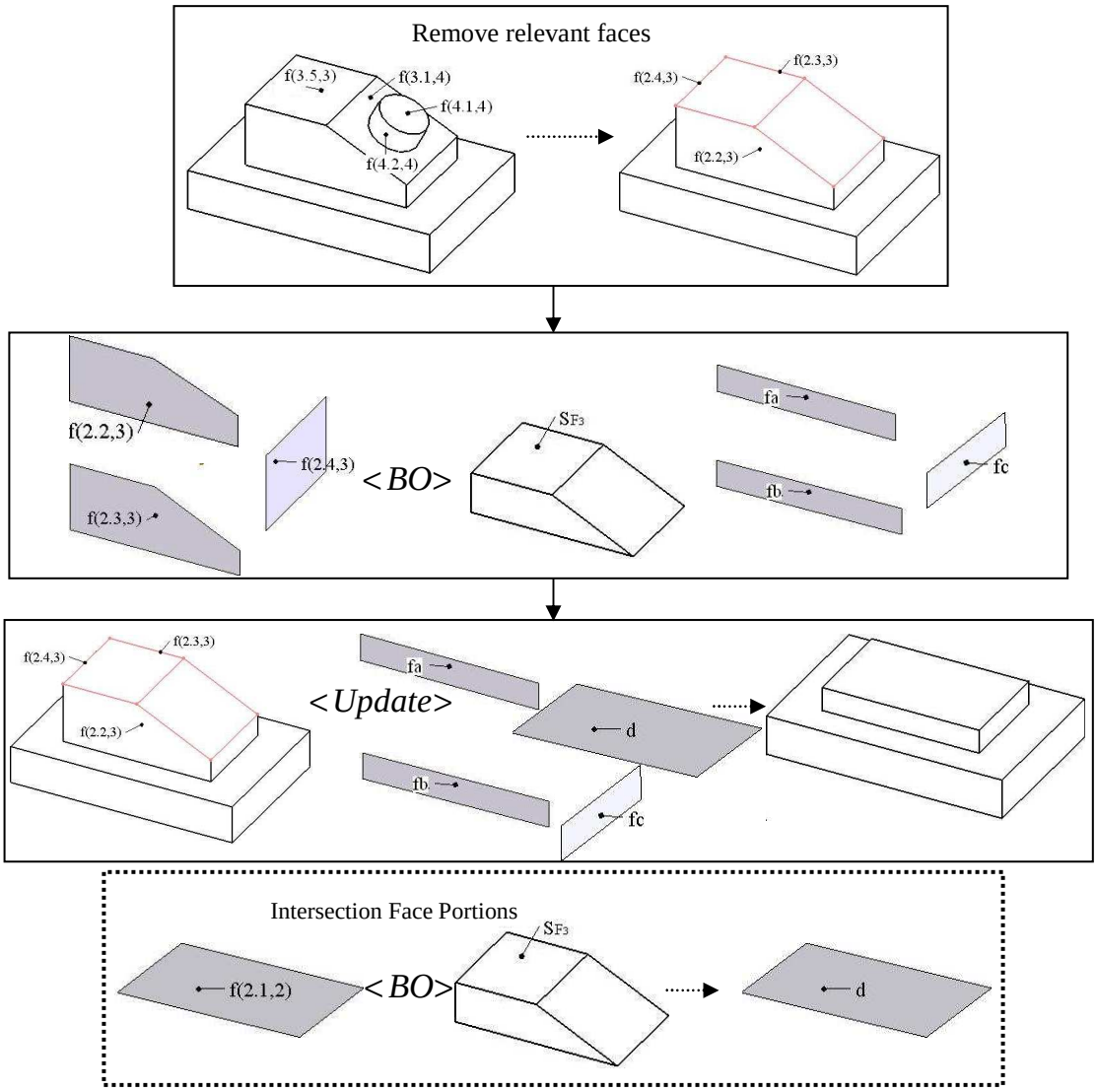


Fig. 3.7 'Remove feature' operation#2

For the examples in Figure 3.6 and Figure 3.7, the intersection face portions stored at the feature creation step can be merged to the model boundary directly. However, when the feature being removed has intersecting features that are created later, the *BCs* of the stored faces should be updated by considering the impact from the intersecting features. As shown in Fig. 3.8, the product model is developed by sequentially

combining three features F_2, F_3, F_4 to the initial feature F_1, and the intersecting list is summarized in Table 3.2. F_2 has three intersecting features, namely, F_1 that is created before F_2, and F_3, F_4 that are created later than F_2.

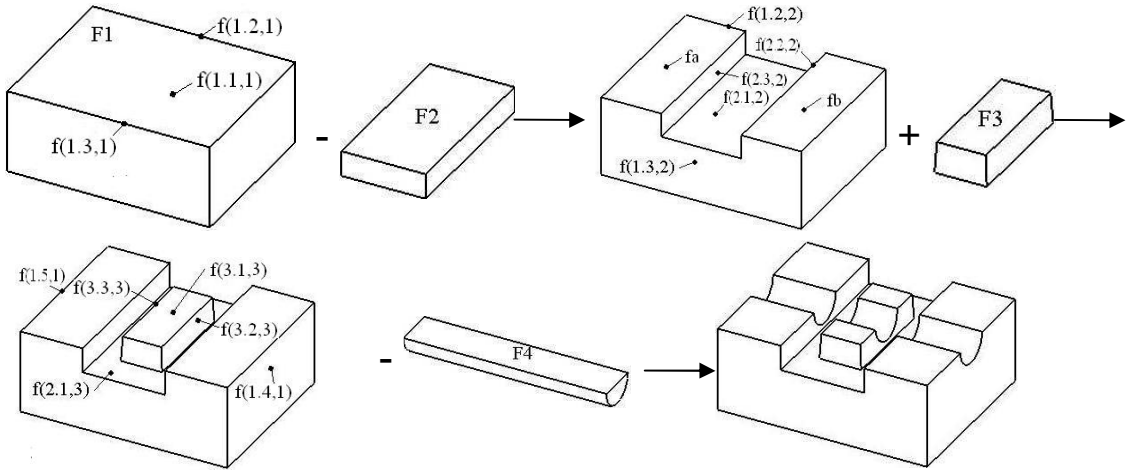


Fig. 3.8 'Add feature' operation#2

Table 3.2 Intersecting list #2

Intersecting List	F_1	F_2	F_3	F_4
Intersecting features	F_2, F_4	F_1, F_3, F_4	F_2, F_4	F_1, F_2, F_3
Intersection face portions		$f_{(1.1,1)} \cap^* F_2$, $f_{(1.2,1)} \cap^* F_2$, $f_{(1.3,1)} \cap^* F_2$	$f_{(2.1,2)} \cap^* F_2$	$f_{(1.4,1)} \cap^* F_4$, $f_{(1.5,1)} \cap^* F_4$, $f_a \cap^* F_4$, $f_b \cap^* F_4$, $f_{(2.2,2)} \cap^* F_4$, $f_{(2.3,2)} \cap^* F_4$, $f_{(3.1,3)} \cap^* F_4$, $f_{(3.2,3)} \cap^* F_4$, $f_{(3.3,3)} \cap^* F_4$

When F_2 is removed from the product model, its attached feature F_3 is removed as well, and the removal operations are shown in Fig. 3.9. At the first step, the boundary faces originating from F_3 are f_c, f_d, $f_{(3.2,4)}$, $f_{(3.3,4)}$, $f_{(3.4,3)}$, $f_{(3.5,3)}$, and the boundary faces originating from F_2 are $f_{(2.1,3)}$, $f_{(2.2,4)}$, $f_{(2.3,4)}$, and they are removed from the model boundary. After removing the faces of F_3, face f_e is disconnected from the remaining model, so it is removed as well. Since F_3 is attached to F_2, its intersection face portion $f_{(2.1,2)} \cap^* F_2$ will not be merged to the model boundary. Hence, at the second step, only

the intersection face portions stored at the creation step of F_2, which are $f_{g,}\, f_h, f_i$, are to be merged to the model boundary. Since F_4 is created later than F_2 and it is intersecting with F_2, the intersection face portions stored at the F_2 creation step may be modified by F_4. In this case, f_i need to be further updated by F_4, which is processed using the Boolean operations with the feature faces defining F_4. After that, the resulting sub-faces and the stored 'intersection face portions' are selectively stitched to the model boundary, which are $f_{g,}\, f_h, f_{i.1}, f_{i.2}$. At the third step, the *BC* of the intersecting feature F_4 is updated, where the Boolean operations between the feature faces defining F_4 and F_2, F_3 are computed. The resulting partitioned faces from the operation with F_3 are f_e and f_f, and the partitioned faces from the operation with F_2 are f_j and f_k. The partitioned sub-faces need to be selectively stitched to or subtracted from the model boundary according to the validity of the resulting BRep model, or the users can be given the opportunity to decide how to process the partitioned sub-faces. In this case, f_k is stitched the resulting model boundary.

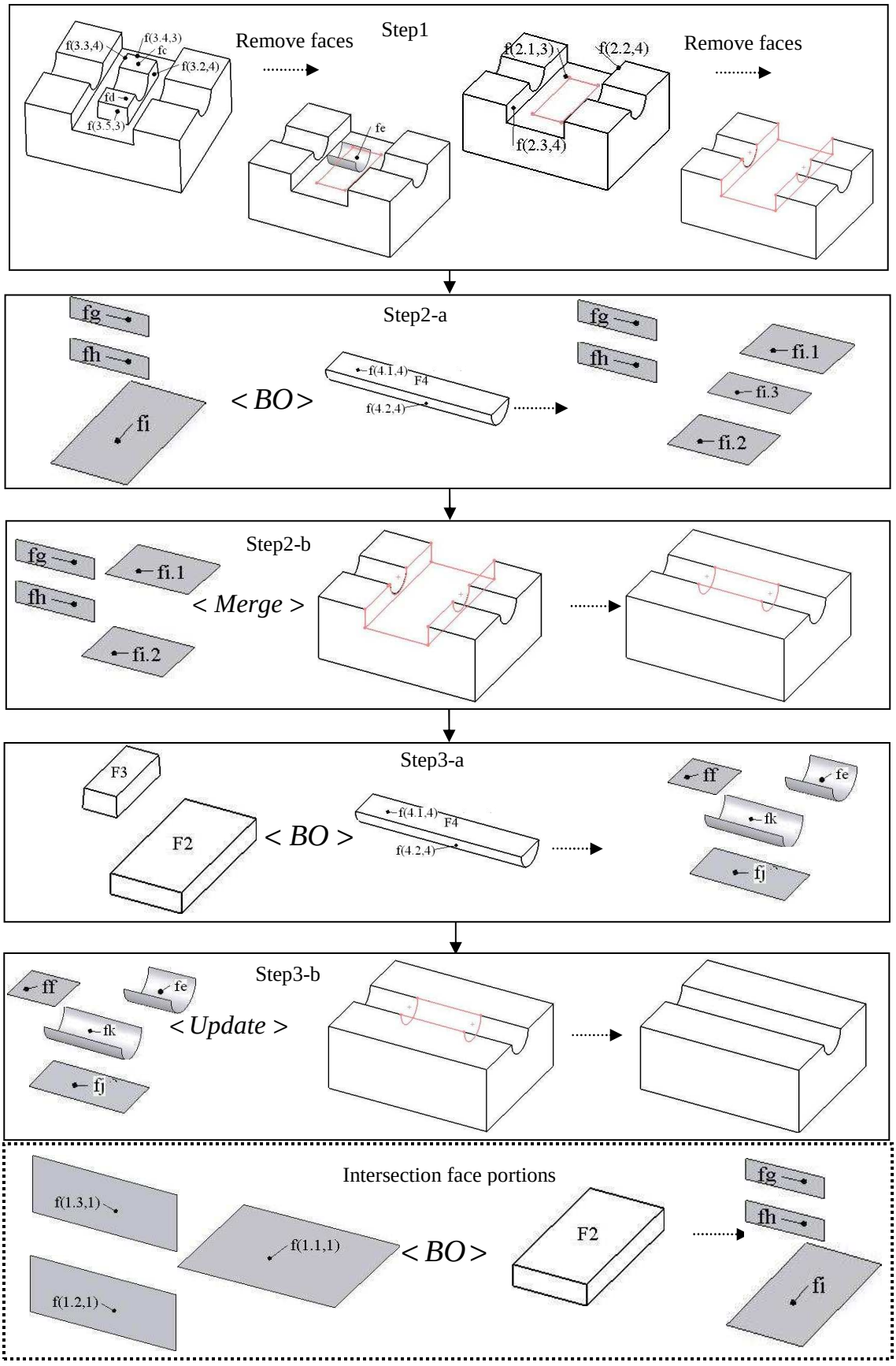


Fig. 3.9 ‘Remove feature’ operation#3

During the 'remove feature' operation, the sub-faces are removed selectively from or merged to the model boundary. The sub-faces originating from the feature being removed should be removed, and the intersection face portions should be merged into the final BRep model, as presented in Figures 3.6-3.7. However, if the feature being removed has an intersecting feature that is created later, the partitioned faces $f_{portion}$, which are generated from the Boolean operations on the intersection face portions and the intersecting features, should be classified with respect to the intersecting feature to determine whether they are on the model boundary, namely $f_{portion}InF_{Inter}$, $f_{portion}OnF_{Inter}$, $f_{portion}OutF_{Inter}$. $f_{portion}OutF_{Inter}$ is merged to the final BRep model, $f_{portion}InF_{Inter}$ is discarded, and $f_{portion}OnF_{Inter}$ is determined based on its 'nature' defined in the specification of F_{Inter}. The 'nature' of a feature face expresses whether it is on the model boundary or not, which is also used in the cellular model representation (Bidarra *et al*., 2005). As the f_i in $Step2-d$ of Fig. 3.9, the partitioned faces $f_{i.1}, f_{i.2}, f_{i.3}$ are classified with respect to F_4. $f_{i.1}, f_{i.2}$ are merged to the final model boundary since they are out of F_4. $f_{i.3}$ is discarded since it is on F_4 and the 'nature' of the feature face indicates it is not on the boundary. For updating the *BCs* of the intersecting features, the sub-faces from the intersecting features are classified by their 'nature' or are classified by the decisions from the users, and in both cases the final BRep model should be maintained valid and consistent. As in $Step3-b$ in Fig. 3.9, f_f and f_j are discarded since their 'nature' in F_4 indicates they are not on the boundary, and f_e and f_k are merged into the final BRep model since their 'nature' in F_4 is on the boundary. In this case, the boundary edges of f_e are not consistent with the intermediate model, and hence f_k is merged to obtain the final BRep model.

3.4.3 'Modify feature' Operation

The 'modify feature' operation is similar to the 'remove feature' operation. When a feature is modified, the features attached to it are usually modified as well. For the 'modify operation', there are two major steps. Firstly, the modified feature and its attached features are removed from the current product model. Next, the modified feature and its attached features are re-added to the product model with newly modified parameters.

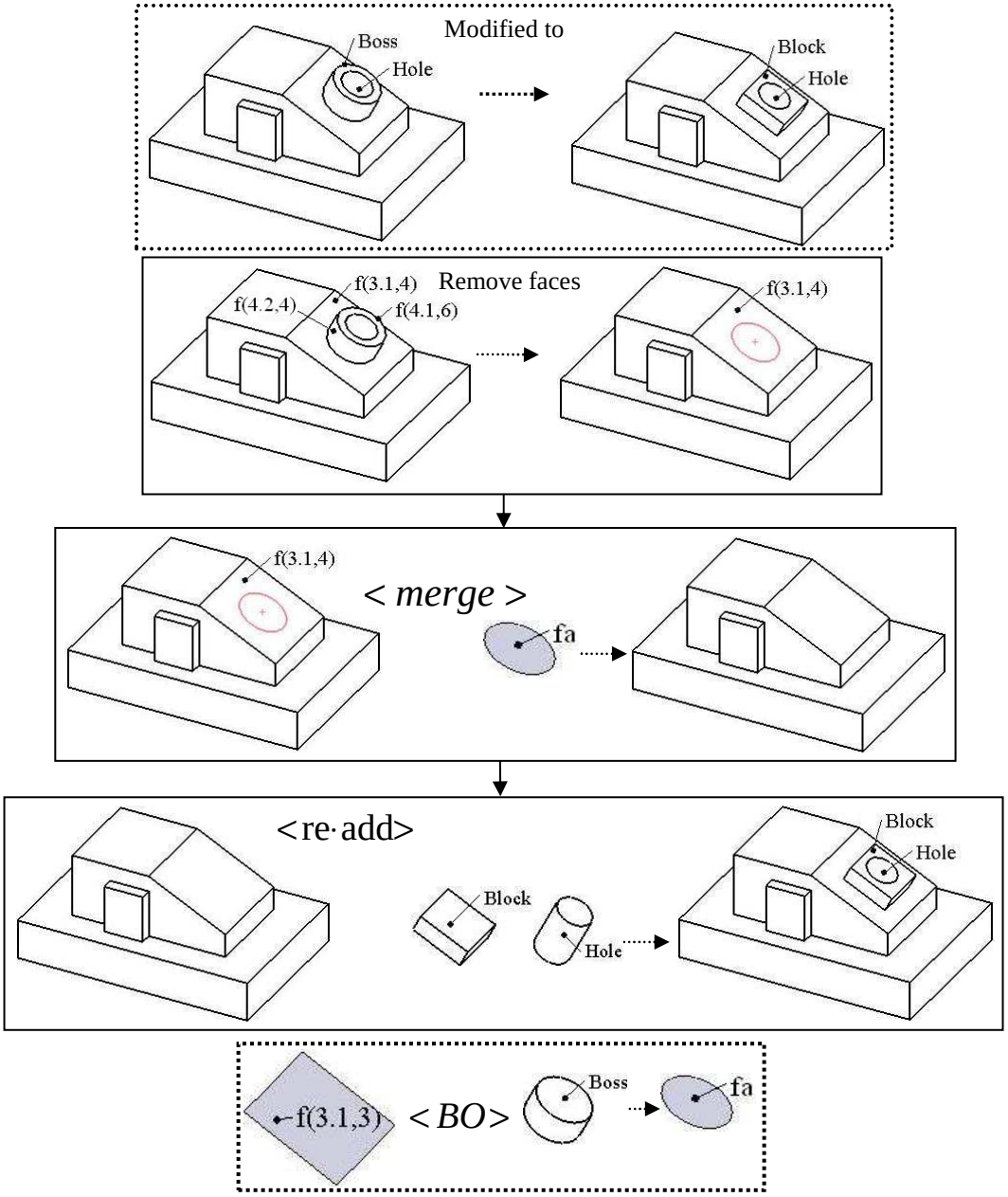


Fig. 3.10 'Modify feature' operation

The ‘removal operations’ and the ‘re-added operations’ have been presented in Section 3.4.2 and Section 3.4.1 respectively. As shown in Fig. 3.10, when the feature *Boss* is modified to become a *Block*, the *Boss* and its attached feature, which is the feature *BlindHole*, are removed from the current product model first. Next, the modified *Block* and *BlindHole* are re-added to the updated product model as the new features.

3.5 Computational Complexity Analysis and Performance Measurement

Three classes of feature models were introduced by Bidarra *et al*. (2005), namely, the best, average, worst case behaviors for analyzing the computational complexity. For the model with the best case behavior, all design features are disjoint. As shown in Fig. 3.11(a), the model consists of a *Block* with one row of 100 non-intersecting *ThroughHole* shapes. For the model with the average case behavior, each of its m features has a small average number i of intersections with other features, i being independent of m. As shown in Fig. 3.11(b), the model consists of a *Block* with a row of 33 feature groups, each of which have three intersecting features inserted sequentially: first *Rib*, then *Slot*, and finally *ThroughHole*. Each *ThroughHole* intersects one face of *Rib*, two faces of *Slot*, as well as one face of the *Block*. For the model with the worst case behavior, each of its m features intersects (once) i other features, and i is smaller than m. The worst model is modified a bit in this work. As shown in Fig. 3.11(c), the model consists of a *Block* with 20 similar *ThroughHole*, which are added in a criss-cross manner. The diameter of the set of holes in one direction is slightly smaller than the diameter of the set of holes in the other direction. Only the computation for the set of holes with the smaller diameter is measured.

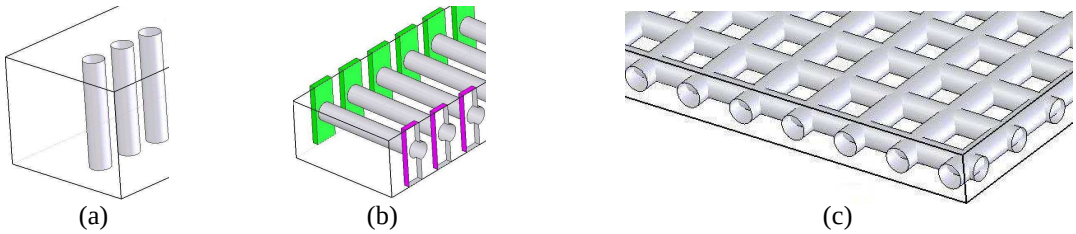


Fig. 3.11 Representative models for (a) best case, (b) average case, (c) worst case behavior (Bidarra *et al*., 2005)

3.5.1 Setup used for measurement

The boundary evaluation for the three representative models in the pure history-based modeling is measured using the commercial software SolidWorks 2006 SP0.0. SolidWorks uses a so-called *swp* file for recording all the user actions, which serves as the script file. The *swp* file can be executed in another modeling session, during which the same commands are executed automatically and the identical resulting BRep model is generated. In order to measure the evaluation time, the time-stamp commands are embedded into the *swp* file so that the start time and the finish time of a modeling command are recorded, as shown in the text box below. On the other hand, the boundary evaluation for the three representative models using the proposed methods are measured based on Open CASCADE. All the performance measurements are carried out in the Windows XP environment on a computer with Intel Duo CPU 2.0GHz and 2G of RAM.

```
boolstatus = Part.Extension.SelectByID2("Top Plane", "PLANE", 0, 0, 0, False, 0, Nothing, 0)
Part.InsertSketch2 True
        QueryPerformanceFrequency curFreq
        QueryPerformanceCounter curStart
boolstatus = Part.Extension.SelectByID2("Sketch1", "SKETCH", 0, 0, 0, False, 0, Nothing, 0)
Part.FeatureManager.FeatureExtrusion2 True, False, False, 0, 0, 0.01, 0.01, False, False, False,
False, 0.01745329251994, 0.01745329251994, False, False, False, False, 1, 1, 1, 0, 0, False
Part.SelectionManager.EnableContourSelection = 0
        QueryPerformanceCounter curEnd
        dblResult(X) = (curEnd - curStart) / curFreq * 1000
```

It is a complex procedure to measure the exact modeling times for the boundary evaluation in SolidWorks since the working algorithms in SolidWorks cannot be accessed and modified easily. Although the *swp* file can record the modeling commands, it is difficult to determine that the measured time is the exact boundary evaluation time. In addition, the specific modeling algorithms in Open CASCADE are very different from that in SolidWorks, and the CPU timers are not accurate. As a result, it is not meaningful to compare the absolute computation times measured in these two modeling software, and the measured times with those reported works in the literature. In this work, all the modeling times are normalized to make them comparable, in which only the trends of the computation times with respect to the number of features are analyzed.

In the 'add feature' operation, the boundary evaluation is measured to find the relationship between the computation time and the number of features added. It is assumed that there are $n-1$ features in the model, and the number n feature is being added to the model. In the 'remove feature' operation and the 'modify feature' operation, the boundary evaluation is measured to find the relationship between the computation time and the sequence position of the feature being edited. It is assumed that there are m features in the model, and the number k feature is being removed or modified.

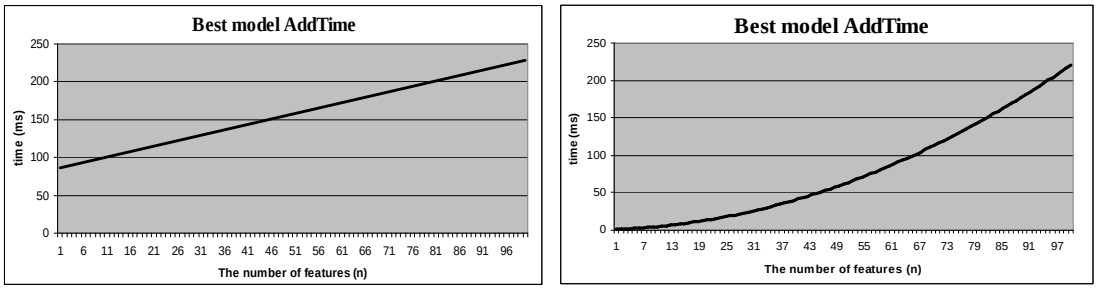
3.5.2 'Add feature' operation

The computational complexity and the performance measurements for the boundary evaluation in history-based modeling have been reported by Bidarra *et al*. (2005). For the 'add feature' operation, the computation time is proportional to the number of boundary faces and the number of intersections per feature. From the charts on the left

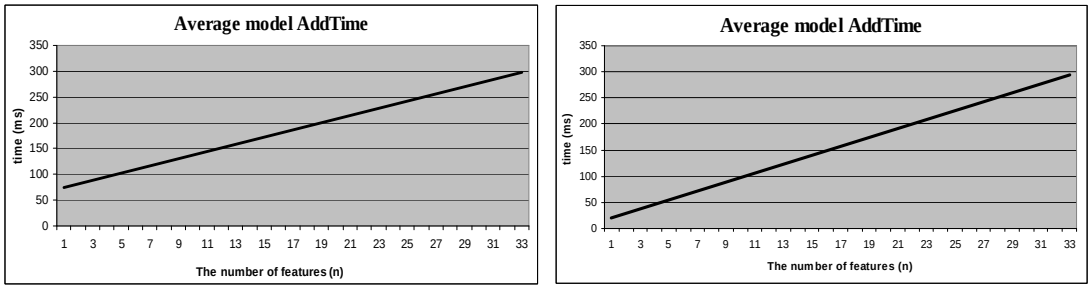
column in Fig. 3.12, for the best case and the average case behavior models, the computation times increase linearly with increasing number of features; for the worst case, the computation time increases in quadratic order with increasing number of features.

In the proposed modeling approach, the 'add feature' operation is similar to the history-based modeling approach. Thus, the computational complexity of the proposed approach is similar to the history-based approach. Bidarra *et al*. (2005) have explained that the computation time for the 'add feature' operation is composed of $t = t_{\text{int}} + t_{op} + t_{upd} = \alpha \times n_{\text{mod}el} + \beta \times n_{\text{int}} + \gamma \times n_{op}$, where t_{int} represents the time required for the identification of the n_{int} intersecting faces from the model boundary that has a total number of $n_{\text{mod}el}$ faces, t_{op} represents the time required for processing these intersecting faces n_{int} using Boolean operations, t_{upd} presents the time required for updating the BRep model with the processed faces n_{op}, and α, β and γ are positive factors. For the 'add feature' operation, the evaluation times of the three representative models are shown in the charts on the right column in Fig. 3.12. For the case of the best behavior model, the computation time increases with increasing number of features in a nearly quadratic order. For the case of the average behavior model, the computation time increases linearly with the number of features. For the case of the worst behavior model, the computation time increases in a cubic order with increasing number of features. It is observed that the computational complexity for the best behavior model and the worst behavior model is one order higher as compared with that in SolidWorks. This is because the number of topological edges in the intersecting faces increases proportionally with increasing number of features, and hence the

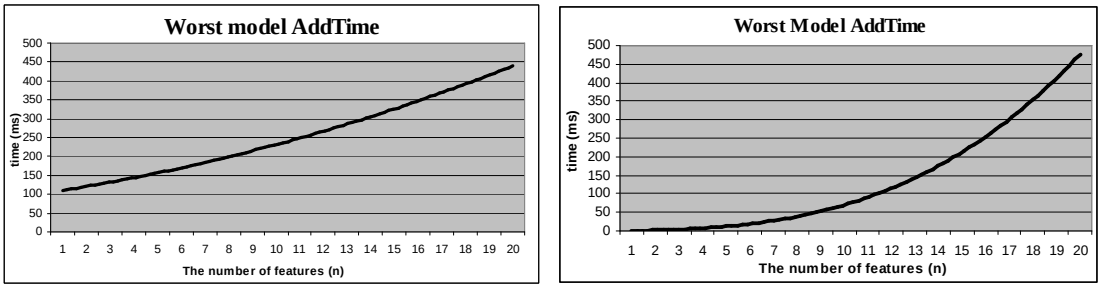
positive factor β becomes larger linearly, which represents the time for performing Boolean operations on the intersecting faces and the new feature faces. The increasing β is due to the specific modeling algorithms in Open CASCADE, and one should adapt the curves of computation times when it is compared with that in SolidWorks and that in other reported works.



(a) best case



(b) average case



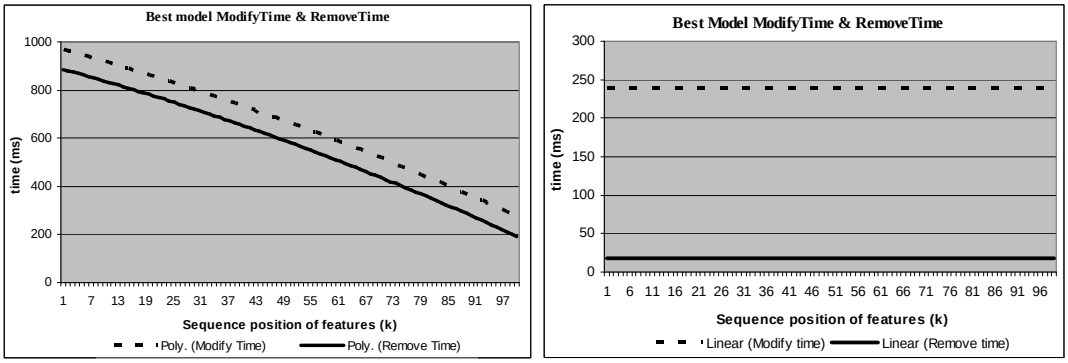
(c) worst case

Fig. 3.12 Measurement of boundary evaluation time for adding a feature using SolidWorks (left column) and using the proposed modeling method (right column)

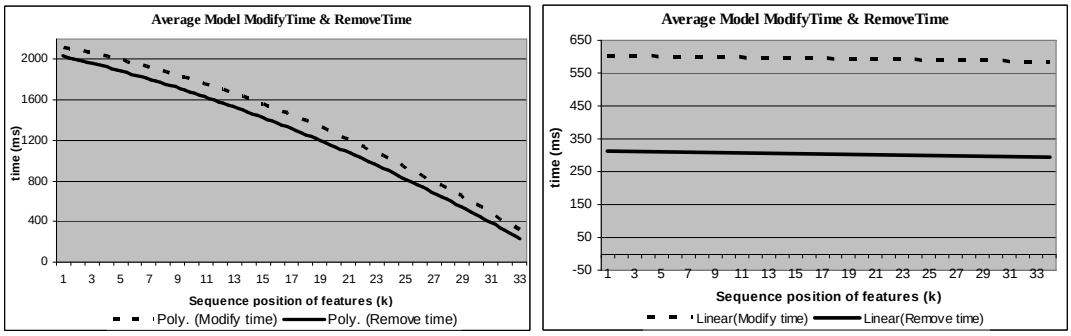
3.5.3 'Remove feature' operation

For the 'remove feature' operations in the pure history-based modeling approach, the computation time is dependent on the sequence position of the feature being removed in the model history. For the charts (solid curves) in the left column in Fig. 3.13, for the best behavior model and the average behavior model, the computation times for

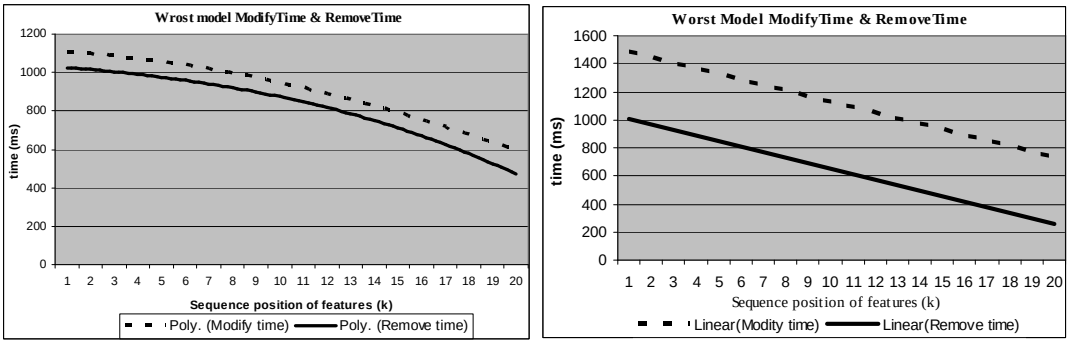
removing a feature decrease in quadratic order when its sequence position in the model history increases; for the worst behavior model, the computation time for removing a feature decreases in cubic order when its sequence position in the model history increases.



(a) best case



(b) average case



(c) worst case

Fig. 3.13 Measurements of boundary evaluation times for removing and modifying a feature using SolidWorks (left column) and using the proposed modeling method

In the proposed boundary evaluation, removing a feature F from a model made up of m features is accomplished through a selective sequence of operations, including performing the deletion, executing a Boolean operation, and merging of the boundary

faces, as explained in Section 3.4.2. Since the 'remove feature' operation is accomplished in three steps, the required computation time can be decomposed into $t = t_{\text{rem}} + t_{\text{StoredFace}} + t_{\text{intFea}}$, where t_{rem} represents the time associated with the removal of the boundary faces originating from F, including removing the relevant topological faces and subtracting the face portions from the relevant merged faces. $t_{\text{StoredFace}}$ represents the time required for processing the faces stored at the F creation step, namely updating and merging the intersection face portions, and t_{intFea} represents the time required for updating the BCs of the intersecting features F_{Inter}.

One can legitimately assume that:

- The boundary face f_F originating from F is deleted from the model boundary directly, and the boundary face f_{merged} is updated by removing the face portions that belong to F. Thus the time t_{rem} is dependent on the number of f_F ($n_{f(F)}$) and the number of f_{merged} ($n_{f(merged)}$).
- The intersection face portion $f_{InterFacePortion}$ is updated in two ways: one is to stitch the face portions directly, and the other is to selectively stitch the resulting sub-faces of the Boolean operation on $f_{InterFacePortion}$ and $f_{InterFeatureB}$ ($f_{InterFeatureB}$ are the faces defining the intersecting feature F_{Inter}) to the model boundary. Thus, the time $t_{\text{StoredFace}}$ is related to the number of the stored face portions: face portions intersecting with $f_{InterFeatureB}$ ($n_{f(StoredFaceA)}$) and face portions without intersecting with $f_{InterFeatureB}$ ($n_{f(StoredFaceB)}$).
- For the BC of the intersecting feature F_{Inter}, the partitioned faces, which are generated by performing $f_{InterFeatureB} < BO > f_{FB}$, are selectively stitched to or

subtracted from the model boundary. Thus, the time t_{intFea} is proportional to the number of the faces that in $f_{InterFeatureB}$ and intersects with F, $n_{f(IntFFace)}$.

The operation behaviors can be expressed by some positive factors, so the former equation can be written as

$$t = \alpha'_1 \times n_{f(F)} + \alpha'_2 \times n_{f(merged)} + \beta'_1 \times n_{f(StoredFaceA)} + \beta'_2 \times n_{f(StoredFaceB)} + \gamma' \times n_{f(IntFFace)}.$$

The computation equations can be analyzed using the representative models in Fig. 3.11 as follows.

Best case. As the model shown in Fig. 3.11 (a), when removing the *kth* hole (h), there is only one boundary face ($n_{f(F)}$ = 1) originating from h and two stored faces ($n_{f(StoredFaceA)}$ = 2). Since h has no intersecting feature that is created later, the time t_{intFea} is zero. The equation representing the required computation time can then be written as $t_b = \alpha'_1 + 2 \times \beta'_1$. From the charts (solid curve) in the right column in Fig. 3.13(a), the computation time for the removal operation remains almost constant as the sequence position of features increases.

Average case. As the model shown in Fig. 3.11(b), when removing the *kth* *Rib*, there are five boundary faces ($n_{f(F)}$ = 3 and $n_{f(merged)}$ = 2) originating from *Rib* and one intersecting face ($n_{f(StoredFaceB)}$ = 1). The intersecting feature created later than *Rib* is a *ThroughHole*. By performing the Boolean operation between *ThroughHole* and *Rib*, there is only one face in the *ThroughHole* intersecting with *Rib* ($n_{f(IntFFace)}$ = 1). The equation representing the required computation time can then be written as $t_a = 3 \times \alpha'_1 + 2 \times \alpha'_2 + \beta'_2 + \gamma'$. From the charts (solid curve) in the right column in Fig.

3.13(b), the removal operation of any *Rib* in the model with the average case behavior has almost constant evaluation time.

Worst case. Assume the *kth* smaller hole (h) is removed from the model shown in Fig. 3.11(c), there are 21 boundary faces ($n_{f(F)} = 21$) originating from h and the number of the intersection face portions is $n_{f(StoredFaceA)} = 2 + 2k$. For the intersecting features created later than h, the number of the faces that intersect with h is $n_{f(IntFFace)} = 20 - k$. The equation representing the required computation time can be written as

$$t_w = 21\times\alpha'_1+(2+2k)\times\beta'_1+(20-k)\times\gamma' = (2\beta'_1-\gamma')k + (21\alpha'_1+2\beta'_1+20\gamma') \quad .$$

Since β'_1 represents the time for performing merging operations of face portions, while γ' represents the time for performing Boolean operations on the faces in the intersecting hole. Hence, $2\beta'_1-\gamma'$ is reasonably assumed to be a negative factor, and t_w decrease linearly with the sequence position k . As the charts (solid curve) in the right column in Fig. 3.13(c), the computation time has a decreasing linear relation with the sequence position of the hole being removed.

3.5.4 'Modify feature' operation

For the 'modify feature' operations in the pure history-based modeling, the computation time is dependent on the sequence position of the feature being modified in the model history. From the charts (dashed curves) in the left column in Fig. 3.13, for the best behavior model and the average behavior model, the computation times for modifying a feature decrease in quadratic order with increasing feature sequence position in the model history; for the worst behavior model, the computation time for modifying a feature decreases in cubic order with increasing feature sequence position in the model history

In the proposed boundary evaluation, modifying a feature F is accomplished by identifying the features affected by the 'modify operation' (F and its attached features), which are removed from the model and re-added with new parameters, as presented in Section 3.4.3. Therefore, the total computation time for a 'modify feature' operation is the sum of the 'remove operation' and the 're-add operation'. In the measurement, it is assumed that the topology of the model is not changed after the 'modify operation'. Since re-adding the feature being modified to the BRep model is similar to adding a new feature to the model, it has the same computation time for any feature being modified. Consequently, the trend of the computation time of the 'modify feature' operation is identical to that of the 'remove feature' operation. From the charts (dashed curves) in the right column in Fig. 3.13, for the case of the best behavior model and average behavior model, the computation time for modifying each hole or rib keeps almost constant with increasing feature sequence position; for the case of the worst behavior model, the time for the modify operation has a decreasing linear relation with the sequence position of the hole being modified.

3.5.5 Analysis and comparison of the performance measurement

Due to the inaccuracy of the computer CPU timers and the different modeling algorithms used, it is not significant to compare the absolute computation times of the boundary evaluation in SolidWorks with that of the proposed modeling method. In addition, in this work, the computation time for performing Boolean operations increases linearly with increasing the number of topological edges in the processed faces. Hence, compared with other modeling solutions, the trend of the computation times in this proposed approach is one order higher in the 'add feature' operations for the best case and the worst case behavior models, as shown in the charts on the right columns of Fig. 3.12(a) and (c).

In the comparison here, the Boolean operations on the intersecting faces are reasonably assumed to be constant regardless of the topological complexity of the intersecting faces, which are similar to the case in SolidWorks and the approach reported by Bidarra *et al*. (2005). Hence, the computation curves of the 'add feature' operations in Fig. 3.12(a) and (c) are decreased one order of complexity. The trends of the boundary evaluation times for the representative models measured using SolidWorks, the proposed approach, and Bidarra's approach are summarized in Table 3.3.

Table 3.3 Trends of boundary evaluations for representative models

Operation	Best case			Average case			Worst case		
	Proposed approach	Bidarra's approach	SolidWorks	Proposed approach	Bidarra's approach	SolidWorks	Proposed approach	Bidarra's approach	SolidWorks
Add feature	Linear ↑	Linear ↑	Linear ↑	Linear ↑	Linear ↑	Linear ↑	Quadratic ↑	Quadratic ↑	Quadratic ↑
Remove feature	Constant	Constant	Quadratic ↓	Constant	Constant	Quadratic ↓	Linear ↓	Constant	Cubic ↓
Modify feature	Constant	Constant	Quadratic ↓	Constant	Constant	Quadratic ↓	Linear ↓	Constant	Cubic ↓

3.6 Case Study

In the proposed approach, each feature has an intersecting list recording its intersecting features and storing its intersection face portions when it is evaluated. The intersecting features are identified when a feature is added to the intermediate BRep model, and hence it does not require any additional computation cost. For the issue of storage space, only the intersection face portions at each feature creation step are stored, and this needs less storage space than the two approaches in Fig. 2.6. However, the proposed approach requires a more mature database management algorithm. The intersecting list needs to be updated instantly, and the alteration process of the feature faces needs to be maintained during the design session. Compared to the approach reported by Bidarra *et al*. (2005), the proposed approach requires more computation time since it does not store all the sub-faces of each feature but only stores the intersection face portions. On the other hand, the database management cost in the

proposed approach is much less than Bidarra's approach. In Bidarra's work, all the *cell* faces and the relevant information are managed, while in this work only the alteration process of the feature face is maintained.

A proof-of-concept prototype system for the proposed modeling approach has been established based on the Open CASCADE. Fig. 3.14 shows a case study model, in which the height of the feature *Rib* is increased. The proposed feature modification approach is employed on this case model to study the proposed modeling approach: firstly the boundary faces originating from the *Rib* are removed, secondly the intersection face portion of *Rib* is merged to the model boundary, and thirdly the new *Rib* is re-added to the update BRep model. The computation time for processing the relevant operation performance is 125ms. In history-based modeling, since the *Rib* is created before the *cirSlot*, the increase of its height would cause the overlapping with the *cirSlot*. In this case study, the height of the *Rib* can be increased and the shape of the *Rib* is not changed by the *cirSlot*, which corresponds to the designer's specification.

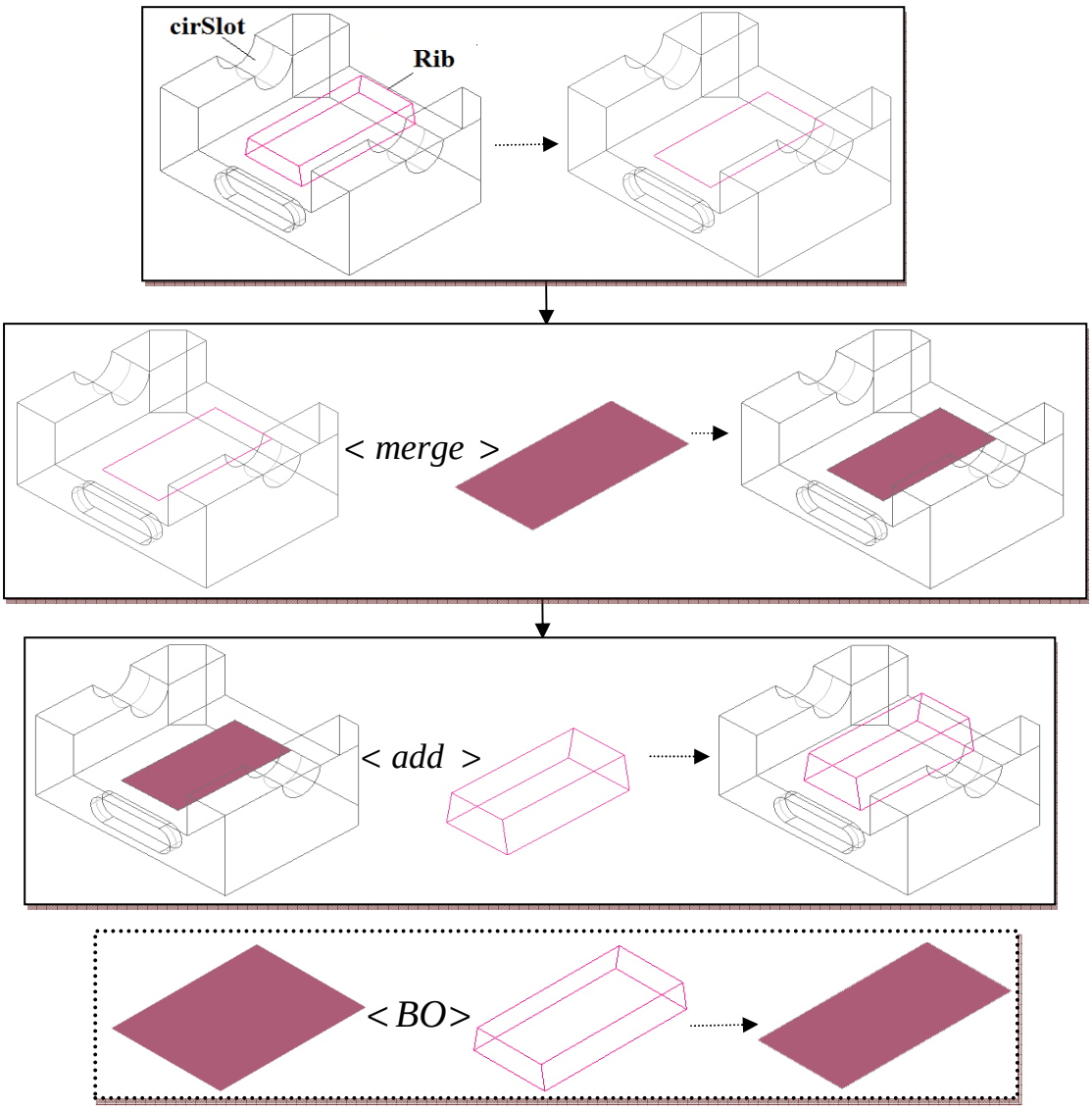


Fig. 3.14 Case study

3.7 Summary

A new history-independent modeling approach has been proposed in this paper, where the feature creation order in the model history can be changed. In this work, to modify a feature is basically to modify the boundary contribution of this feature and the intersecting features, so that the resulting BRep model is updated to reflect this modification. The intersection face portions of a feature being added to the product model are stored at the creation step, and the intersecting features that are created later in the model history are recorded. When a feature is removed, firstly the boundary

faces originating from this feature are removed, secondly the intersection face portions stored at its creation step are merged to the model boundary, and lastly the boundary contributions of its intersecting features are modified. When a feature is modified, it is first removed from the product model and then re-added with the new specifications. Hence, the creation step of the feature being modified is changed, and the problems caused by the static feature creation order are solved. The complexity analysis and performance of the proposed boundary evaluation for three representative models have been analyzed and measured. For case of the best behavior model, the computation time for removing or modifying a hole is almost constant. For the case of the average behavior model, the computation for removing or modifying each rib has almost constant evaluation time. For the case of the worst behavior model, the computation for removing or modifying a hole decreases linearly with the sequence position of the hole in the model history. The case study in a proof-of-concept prototype system demonstrates the feasibility of the history-independent modeling approach.

We recall the problems introduced in history-based modeling as that presented in Fig. 2.2 and Fig. 2.3, which attributes to the follows: the boundary evaluation of the feature being modified is on the basis of the intermediate BRep model in the design history, but the designers modify the feature on the basis of the current BRep model. The proposed modeling solution addressed this problem by changing the order of feature creation operations, which is then consistent with design actions. As shown in Fig. 2.2, when *BHole* is modified and re-positioned, it is removed from the model and then re-added as a new feature to the updated model. Hence, the feature *Rib* can be referred for positioning, since the *Rib* is stored before the *BHole* in the new model history. As shown in Fig. 2.3, the *THole* will intersect with the *Block* , since the *THole* is

evaluated as a new feature and the *Block* is checked for face intersections during the boundary re-evaluation of *THole*.

There are three points in this work that should be highlighted. Firstly, in the 'remove feature' operation, the boundary faces originating from the feature being removed should be identified and removed from the model boundary. Since the feature faces may be trimmed or merged, it is crucial to maintain the alteration process of feature faces, as that presented in Fig. 3.4. In this case, the naming and retrieval of the altering faces must be effective. Once the altered faces that originate from the feature being removed can be identified effectively, the 'remove feature' operation will work smoothly. Secondly, compared to the approach reported by Bidarra *et al*. (2005), the proposed approach requires more computation time in the feature removal and modify operations, but it requires less implementation work for database management. Thirdly, one unresolved issue in the current work is position referencing between features. In this work, when a feature is being removed or modified, only the boundary evaluation is concerned. In real modeling applications, the designers should re-position the child features if their parent feature has been removed or modified. As a result, the function for re-position of child features will be explored in future work, such as providing the designers with several optional topological entities.

Chapter 4 Coordination in Replicated Collaborative Feature Modeling

4.1 Introduction

The coordination mechanism for scheduling the collaborative design activities has been a topic of significant research effort. Reported studies, namely, total-locking mechanism and granular locking mechanism, have been presented in the review Section 2.2.2.1. The reviews show that the granular locking mechanisms have some limitations. Although the designers can edit different parts of a feature model at the same time, there are some conflicts due to feature interactions. Besides, in order to maintain consistency of the replicated design models, the order of execution of the 'feature create operations' must be kept consistent at the client sides.

In this Chapter, a fine granular locking mechanism is presented for a replicated collaboration system. The locking granularity is defined according to feature relationships, and the potential operation conflicts are resolved using a naming and matching mechanism. In the proposed approach, a design model can be divided into several feature portions, thus a parallel working paradigm can be achieved. The 'feature create operation' is processed differently from the 'feature modify operation' so as to maintain consistency of the order of the features created. The limitation of this approach is that the effectiveness depends on the parent-child relations of the features. The proposed coordination approach has been validated in a proof-of-concept prototype system developed based on Java and Open CASCADE. The remaining content of this Chapter is organized as follows. Firstly, the proposed granular locking mechanism is presented. Secondly, the methods of conflict resolution are elaborated. Thirdly, the proposed granular locking mechanism is validated using a case study.

4.2 Granular Locking Mechanism

In feature-based design, a product model is basically a combination of a group of specific features. In this case, the basic elements in the product model are feature shapes rather than geometric elements, e.g., curves, surfaces, so the design activity is more efficient than that in geometric modeling. In addition, since the features in a product model have parent-child relations in their parametric definition, the manipulation of a feature usually affects its direct ancestral and direct descendent features. Consequently, in this work, features in a model are grouped to define the locking granularity, and hence the design operations that are performed on the features are classified accordingly.

4.2.1 Feature Dependency Relationship

If a feature F_1 is attached, positioned or constrained relative to the boundary entities of another feature F_2, then F_1 depends on F_2, F_2 is the direct ancestor of F_1, and F_1 is the direct descendant of F_2. The dependency scope (DS) of a feature F is denoted in Eq. (4.1). If the DSs of two features being edited do not share any common feature, the DSs are mutually exclusive and the two operations are unrelated operations, which can be executed concurrently at different client sites. Otherwise, if the DSs of the two features being edited overlap, then the two operations are dependent operations, and should not be executed concurrently. As shown in Fig. 4.1, the *Rib* is attached and positioned to the initial *Stock* , and the *BlindSlot* is attached and positioned to *Step* . Hence, $DS(Rib)$ and $DS(BlindSlot)$ are mutually exclusive, and the operations on *Rib* and *BlindSlot* are unrelated operations that can be executed concurrently at different sites.

$$DS(F) = F \bigcup DirectAncestor(F) \bigcup DirectDescendant(F) \qquad (4.1)$$

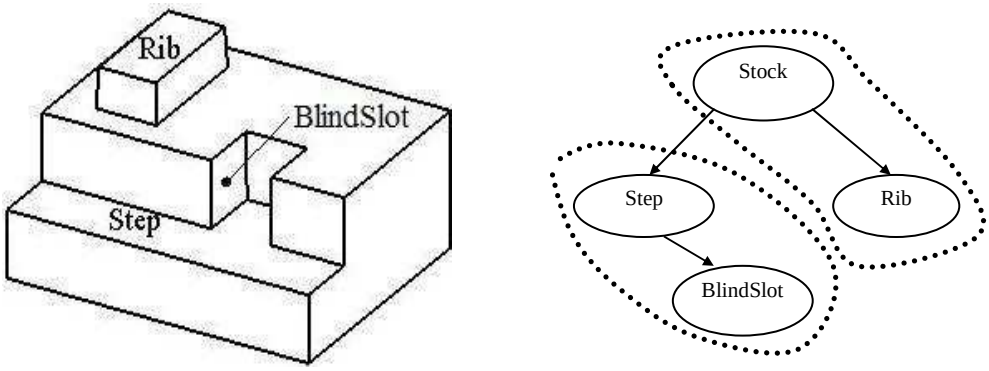


Fig. 4.1 Feature relationships

4.2.2 Concurrency Control

Ordering events occurred in a distributed environment according to their occurrence time is not a new problem, and the general solution reported by Lamport (1978) is adapted in this work. The events occurred at one site has an exact sequence according to the occurrence time. However, in a distributed environment, it becomes a challenge to identify the sequence of the events. In order to determine the sequence of the events in a distributed system, partial ordering and total ordering relations have been introduced. The partial ordering relation, denoted by "$\rightarrow$", satisfies the following conditions:

1. If a and b are events at the same site, and a comes before b, then $a \rightarrow b$.
2. If a is the sending message at one site, and b is the receiving of the same message at another site, then $a \rightarrow b$.
3. If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$.

However, according to partial ordering, concurrent events cannot be ordered. For instance, a is the sending message at one site, and b is the event at another site that occurs just before receiving the message a. It is difficult to determine which event comes first in terms of the exact occurrence time, and they are termed concurrent

events, denoted by $a // b$. In this case, the total ordering relation is introduced, denoted by "$\Rightarrow$", satisfying the following conditions. Each design site has an identity number $S_i (1 \le i \le N)$ that corresponds to its entrance order to the design session, and S_i has a higher priority than S_j if $i < j$, denoted as $S_i \prec S_j$.

1. If $a \rightarrow b$, then $a \Rightarrow b$.
2. If $a_{S_i} // b_{S_j}$ and $S_i \prec S_j$, then $a_{S_i} \Rightarrow b_{S_j}$.

The adapted concurrency control approach employed in this research can be briefly described as follows: when a designer wants to execute a 'modify operation' or a 'create operation' of a feature, this operation can only be performed until the locking of the *DS* of the feature being edited is permitted by all the other designers. At any time, more than one 'modify operation' can be processed concurrently as long as the *DSs* of the features being modified are mutually exclusive, but only one 'create operation' is processed.

In this work, a working session is used for the management of the collaborative design tasks (Li and Qiu, 2006), in which all the client sites are connected to form a distributed and collaborative working environment. Several reasonable assumptions were made to simplify the system introduced in this work: firstly, the number of client sites is fixed and each client is aware of the existence of the rest; secondly, the communication between clients is reliable; thirdly, it is assumed that there is no sudden crash of a client site and no exiting of a client without notice. In the design system, each client site has an integer object *replyCount* , which counts all the replies of an operation request, and an integer object *MS* indicating the current state of the local geometric model. Since the sequence of the 'feature create operations' at each site is

maintained to be identical, *MS* only records the number of features. On the design model, each feature has a unique identity *featureId* and a *requestQ* that records all the locking requests.

4.2.2.1 Modify a Feature. When a designer wants to modify a feature F, the request for locking $DS(F)$ and the 'modify information' *MFInf* are sent to all the other designers, denoted in Eq. (4.2). If any feature in $DS(F)$ is being locked by another designer, the sending of the 'modify operation' request is deferred until all the features in $DS(F)$ are released. At any remote site S_j, after receiving the $RT(MF)$, the system at site S_j uses algorithm#1 to determine the action needed for the modification request received.

Algorithm#1:

If ($requestQ(DS(F))->Contain(S_j) == TRUE$)

{ $if\ (S_i \prec S_j)$ { $reply\ immediatel\ y$; $update\ requestQ$;}

$else\ deferred\ \ Until\ (requestQ(DS(F))->Contain(S_j) == FALSE)$ }

$else$ { $reply\ immediatel\ y$; $update\ requestQ$;}

After receiving all the replies of the $RT(MF)$ from the other designers, the 'modify operation' is first executed at site S_i, after which it is executed at all the other sites to update the replicated design models. The designer/system at any remote site S_j sends a message to the designer/system at site S_i after the execution of the modification.

After receiving all the messages on the successful execution, the *requestQ* of the $DS(F)$ is updated at all the designer sites.

$$RT(MF) < S_i, DS(F), MFInf, MS_{S_i} > \quad (4.2)$$

4.2.2.2 Create a Feature. Since the 'feature creation order' at every site must be kept consistent, only one designer is permitted to create a new feature F at any time. If a designer has been sent a 'feature create operation' request, he will have to wait for this request to be performed first, and therefore his creation request is deferred. Otherwise, a creation request $RT(CF)$ is sent to all the other designers for review and permission, and this includes the locking request of the reference features $DS(F)$ and the 'feature information' *CFInf* , denoted in Eq. (4.3). The 'feature create operation' can only be executed after receiving the permission from the other designers. Since the 'feature create operation' needs to lock certain reference features, the sending of the 'feature create operation' request is deferred until all the reference features are released.

$$RT(CF) < S_i, DS(F), CFInf, MS_{S_i} > \quad (4.3)$$

At any remote site S_j, after receiving the $RT(CF)$, the request is processed according to the following steps and illustrated in algorithm#2.

1. The features being requested are locked by the local designer. $RT(CF)$ is deferred if $S_j \prec S_i$, otherwise it is replied immediately.
2. The features being requested are not locked by the local designer, but a creation request has been sent out by the local designer just before receiving $RT(CF)$. If the two 'feature create operations' are conflicting, these two designers will discuss to decide the execution order of the two operations. Otherwise, $RT(CF)$ is deferred if $S_j \prec S_i$ and is replied immediately if $S_i \prec S_j$.

Algorithm#2:

If ($requestQ(DS(F))->Contain(S_j) == TRUE$)
{ *if* $(S_i \prec S_j)$ { *reply immediatel y* ; *update requestQ* ;}
else deferred Until $(requestQ(DS(F))->Contain(S_j) == FALSE)$ }
else { *if* $(operationConflict)$ *discussSession*;
else { *if* $(S_i \prec S_j)$ { *reply immediatel y* ; *update requestQ* ;}
else deferred Until $(MS_{S_j} += 1)$; }}

After receiving all the replies of the $RT(CF)$, the 'feature create operation' is first executed at site S_i, and then it is executed at all the other sites for updating the replicated design models. The designer/system at any remote site S_j replies a message to the designer/system at site S_i after the execution of the 'feature create operation'. After receiving all the replies of the successful execution, the *requestQ* of the $DS(F)$ and the *MS* are updated at all the design sites.

4.2.3 Correctness analysis of the proposed approach

In a distributed collaborative design environment, *MSs* at the client sites may be different due to the different transmission times (Li *et al*. 2008b). As shown in Fig. 4.2, the 'modify operation' of the *cirSlot* comes before the 'feature create operation' of the *cirSlot* at *Site*3, which is considered a causal conflict. In this case, the 'modify operation' of *cirSlot* is performed when *cirSlot* has not been created, so the causal relation is violated here. In the proposed concurrency approach, the designer needs to send a message to the original operation initiator after the successful execution of an operation so as to avoid the causal conflict. As shown in Fig. 4.2, before sending the modification request *cirSlot* at *Site*2, the $DS(cirSlot)$ must be released. As long as the 'feature create operation' of the *cirSlot* has not been executed at *Site*3, the $DS(cirSlot)$ will not be released. In addition, a deadlock is a tricky problem in the

locking scheme, as presented by Li *et al*. (2008b). In this work, the locking granularity is the *DS* of a feature, which can only be held by one designer at any time. Thus, it is impossible for one designer to hold the *DS* of a feature and wait for another. Hence, the system is deadlock free.

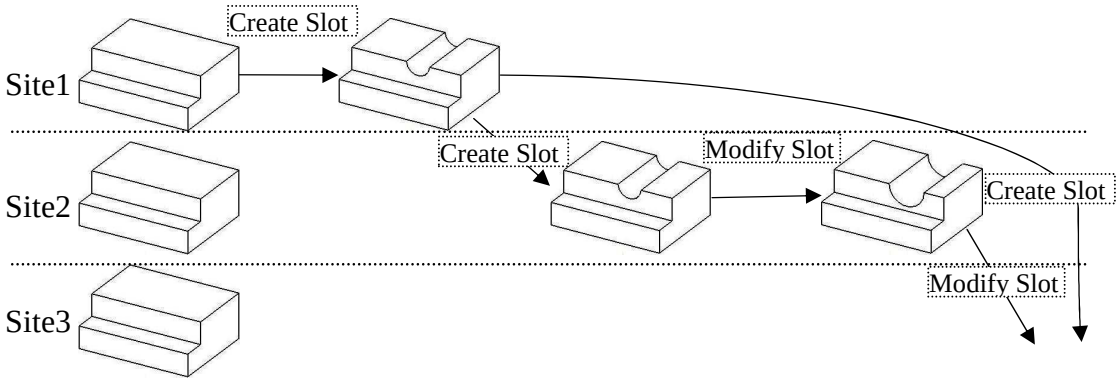


Fig. 4.2 Causal conflict

4.3 Resolution of Potential Operation Conflict

The proposed granular locking mechanism provides a parallel working paradigm, but there are also some potential operation conflicts due to feature interactions. Before any feature F is edited, $DS(F)$ needs to be permitted by the other designers. However, due to feature interactions, the features outside of $DS(F)$ may be interacted by the features in $DS(F)$. As shown in Fig. 4.3, due to the change of position parameter a, *cirSlot* and *rectSlot* become interacting. $DS(cirSlot)$ includes the initial *Stock* and *cirSlot*, and $DS(Rib)$ includes feature *rectSlot* and *Rib*. Since $DS(cirSlot)$ and $DS(Rib)$ are mutually exclusive, *cirSlot* and *Rib* can be modified by two designers concurrently. Assume that *cirSlot* is modified at site S_i, and *Rib* is modified or created at site S_j. Since *Rib* is constrained to edge e_1 for its height and is attached to face f_1, the changes of e_1 and f_1 affect the execution of *Rib* operation. As shown in Fig. 4.3, due to the interaction between *cirSlot* and *rectSlot*, edge e_1 coincides with

edge e_2 in (b) and diminishes in (c), and face f_1 is trimmed in (d). All the topological changes should be resolved in order to execute the operations correctly.

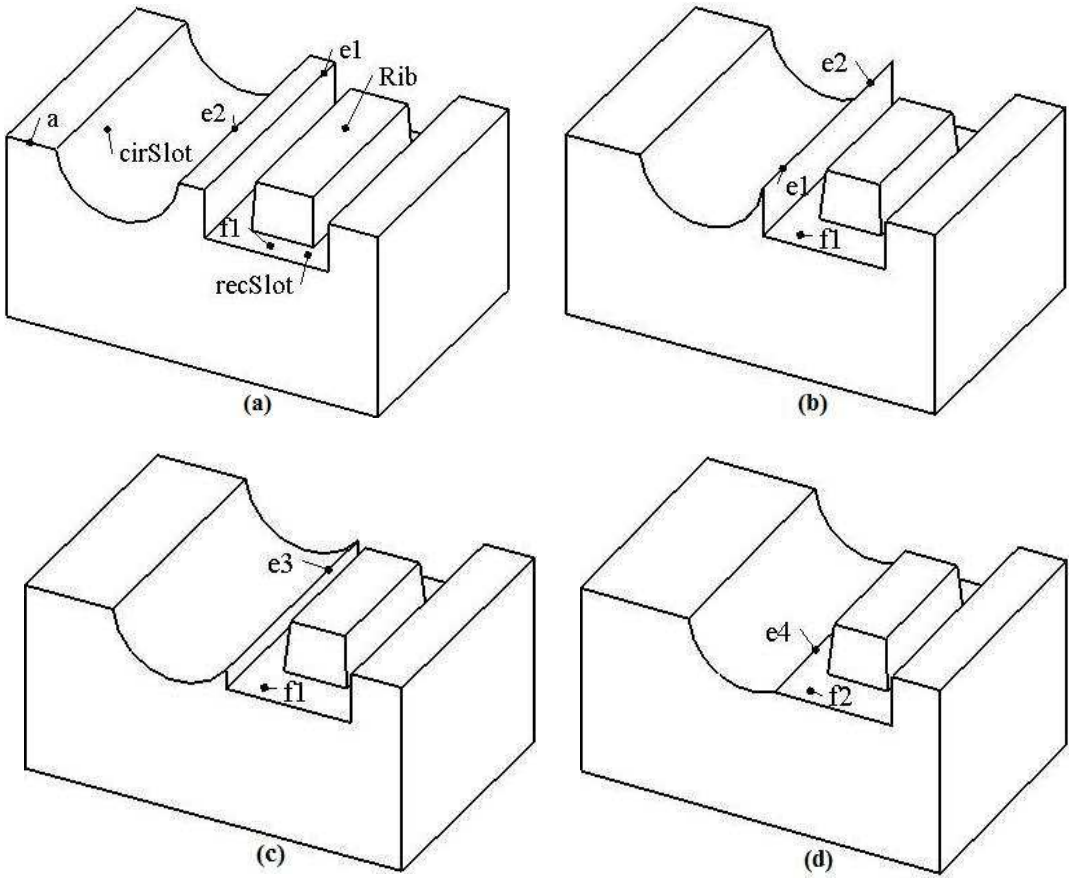


Fig. 4.3 Potential operation conflicts

In a feature operation, the feature is attached, positioned and constrained to some topological entities of the design model, which are termed the reference entities and denoted in Eq. (4.4). As shown in Fig. 4.4(b), a *Boss* is attached to the top face f of a *Block*, and positioned to the edges e_1 and e_2. The reference entities are used to set the location and dimension of the feature. Hence, as long as the reference entities are identified on the design model or some alternative entities can be used to constrain the feature, the operation can be executed correctly. For an entity-based operation, e.g., blending an edge, the execution of the operation needs to identify the 'target entities' as denoted in Eq. (4.5). In Fig. 4.4(c), the edge e_3 is chamfered and thus the entities of

the operation target need to be identified on the design model. As a consequence, in order to resolve the potential operation conflicts, the reference topological entities must be identified on the design model correctly.

$$O(F) = \{ featureId, shapeParameters, referenceEntities \} \quad (4.4)$$

$$O(F) = \{ featureId, parameters, operationEntities \} \quad (4.5)$$

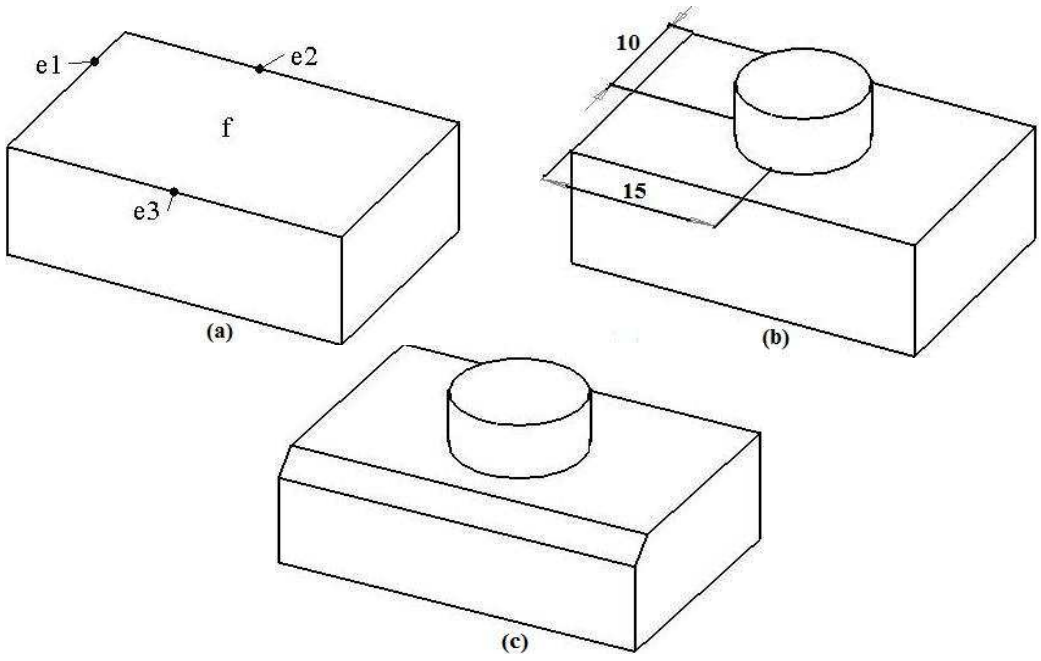


Fig. 4.4 Reference entity in feature operation

4.3.1 Identify Attached Face

In replicated collaborative design, the attached face in an operation performed at one design site may be modified by another operation performed at another site, i.e., the face could have been trimmed, split or merged. As shown in Fig. 4.5(a-b), a *Rib* is created at $Site1$, which is attached to the top face f_1 of the *Stock*. At the same time, since $DS(BlindSlot)$ is the *Step*, the *BlindSlot* can be modified at $Site2$, as shown in Fig. 4.5(c-e). In this case, when the *Rib* 'feature create operation' is broadcast and executed at $Site2$, the face that it is attached to needs to be identified. The steps are as follows.

- When an attached face is trimmed, it is replaced by the trimmed face. As shown in Fig. 4.5(c), although the face f_1 is trimmed, the updated f_1 can be used to locate the *Rib* to the original surface.
- When an attached face is split, each sub-face can be used as the attached face. As shown in Fig. 4.5(d), the face f_1 is split into $f_{1.1}$ and $f_{1.2}$, either of them can be used to locate the *Rib* on the same surface.
- When an attached face is merged, it is replaced by the merged face. As shown in Fig. 4.5(e), the *BlindSlot* is modified to be an extruded *Block*, the merged face f_1 can be used to locate the *Rib* from the original intended surface.

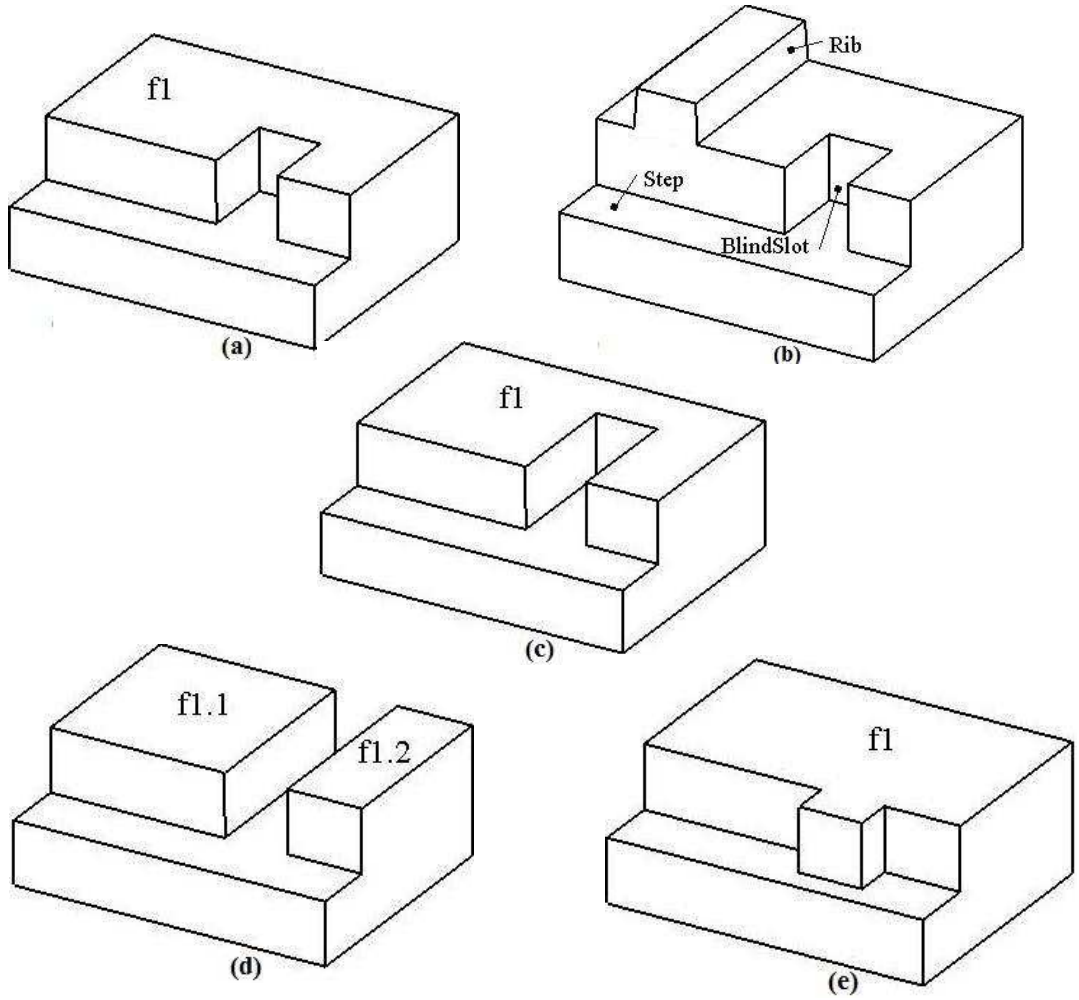


Fig. 4.5 Attaching face alteration

In feature-based design, each feature face can be assigned a unique name in terms of the feature's generating mode and its location in the feature shape (Capoyleas 1996; Wu *et al*. 2001; Wang and Nnaji 2005). As shown in Fig. 4.6, the profile-based feature shape has the profile entities $< e_1, e_2, e_3, e_4 >$ and the sweeping path L, both of which are recorded persistently during the design process. Hence, the side faces of the swept feature shape can be unambiguously named by the profile entities and the sweeping path, and the start face and the end face are named by their specific locations in the feature shape. Combining with the *FeatureId* , all the feature faces on the design model are named persistently, termed the invariant name (*IN*). During the design process, the faces may be trimmed, split or merged.

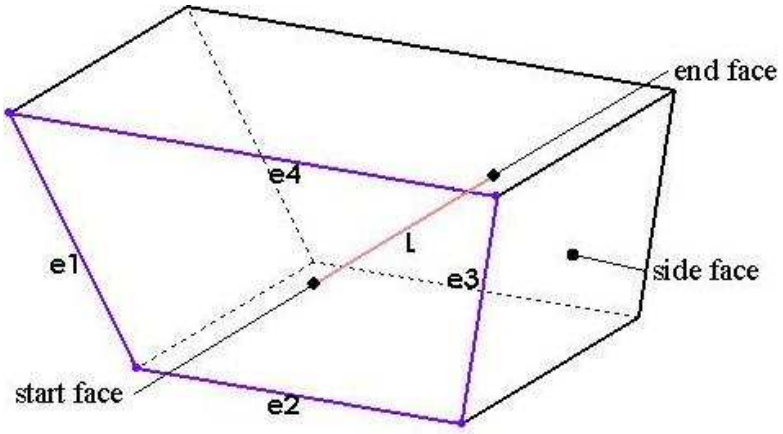


Fig. 4.6 Naming scheme

In order to identify the modified attached faces, the alterations of the boundary faces on the design model are tracked through a *FaceIdGraph*. In the *FaceIdGraph*, each face is assigned a *FaceId* , in which the first item is the *IN* of face and the second item is the *StepId* of this operation, as denoted in Eq. (4.6). When a face is split, the sub-faces can be discriminated in terms of the bounding feature faces and the geometric information of the intersection edges (Cripac 1997; Wang and Nnaji 2005), denoted in Eq. (4.7). Hence, the alternation of the boundary faces can be tracked through

the *FaceIdGraph*, as shown in Fig. 4.7, which shows the face alterations of the model in Fig. 4.5(d).

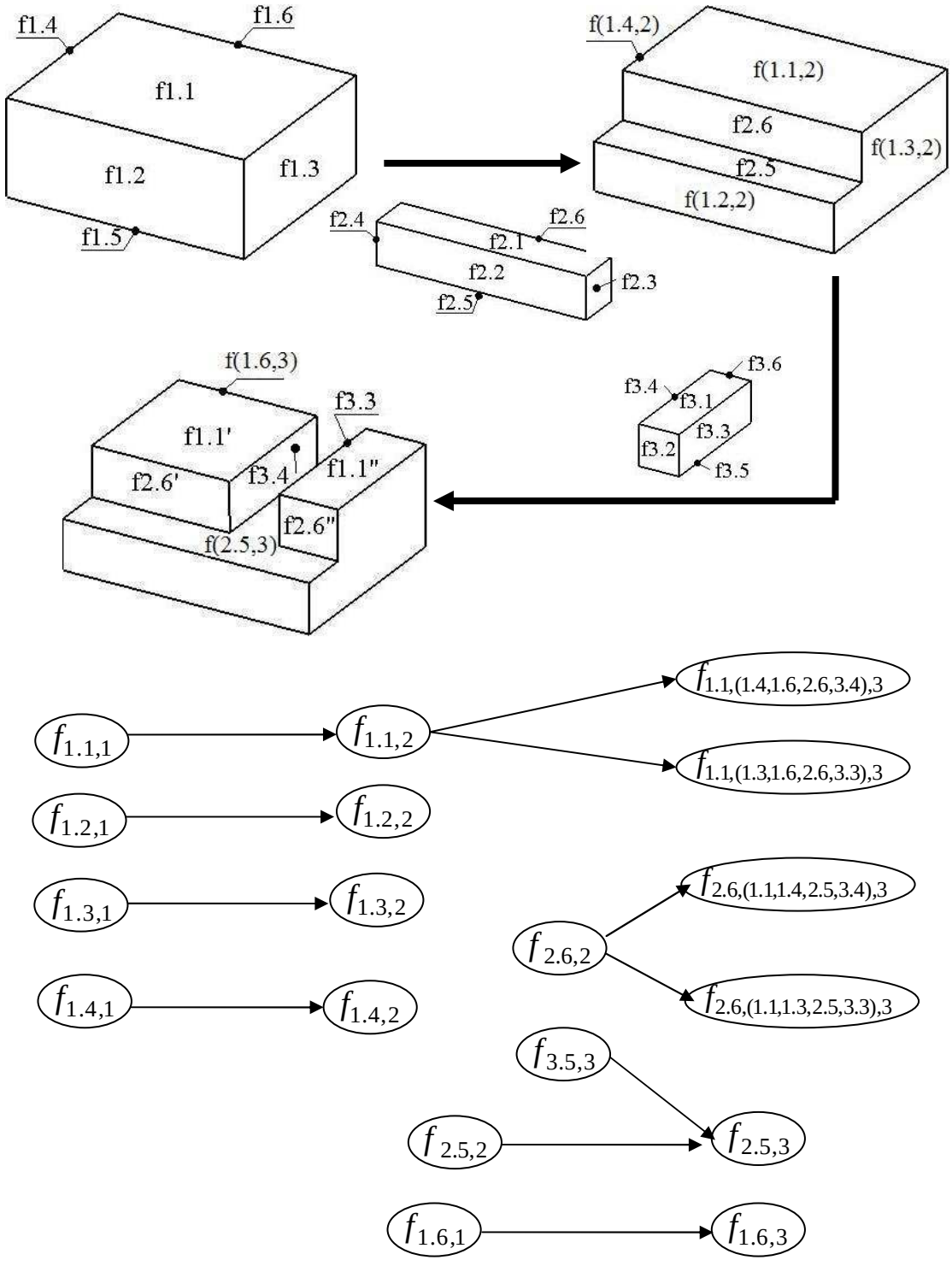


Fig. 4.7 Boundary face alteration tracking

$$FaceId(f) = (IN, stepId) \tag{4.6}$$

$$FaceId(f) = (IN, INs(boundingFaces), stepId) \quad (4.7)$$

$$N(e) = (INs(f_{adj}), INs(f_{bound}), GI(e)) \quad (4.8)$$

$$N(e_1) = ([IN(f_1), IN(f_2)], [IN(f_3), IN(f_4)], GI(a)),$$
$$N(e_2) = ([IN(f_1), IN(f_2)], [IN(f_3), IN(f_4)], GI(b)) \quad (4.9)$$

4.3.2 Identify Reference Edge

The reference edges in an operation performed at one site may be modified by the operation performed at another site, i.e., the edges could have been deleted, split, trimmed or merged. As shown in Fig. 4.8(a-c), a *cirSlot* and a *recSlot* have been attached to the initial *Stock*, and a new feature *Rib* is attached to the *recSlot* at *Site*1, where the topological edge *e* is used as the reference edge. At the same time, since $DS(Rib)$ and $DS(cirSlot)$ are mutually exclusive, the *cirSlot* can be modified at *Site*2. Due to the interaction between *cirSlot* and *recSlot*, the edge *e* is changed. As shown in Fig. 4.8, edge *e* is deleted in Fig. 4.8(d) due to the position change of *cirSlot*. Edge *e* is split in Fig. 4.8(e) and trimmed in Fig. 4.8(f) due to the orientation change of the *cirSlot*. Edge *e* is merged in Fig. 4.8(g) due to the change in *cirSlot*. In this case, when the *Rib* operation is broadcast to *Site*2, the reference edge *e* needs to be identified on the local design model. The reference edge can be named uniquely by its adjacent feature faces, the bounding feature faces and the geometric information *GI* of the edge itself (Wang and Nnaji 2005), as denoted in Eq. (4.8). As shown in Fig. 4.9, the two intersection edges e_1 and e_2 can be named and differentiated using Eq. (4.9). Since the feature faces are named uniquely and stored persistently, their intersection edge can be identified or re-constructed on the design model. As shown in Fig. 4.8(d-g), as long as the *Stock* and the *recSlot* are on the design model, the intersection edge of the face $f1$ and the face $f2$ can be re-constructed in Fig. 4.8(d)

and identified in Fig. 4.8(e, f, g). Hence, the *Rib* operation can be executed correctly. Likewise, the reference vertex can be named and identified in the same way.

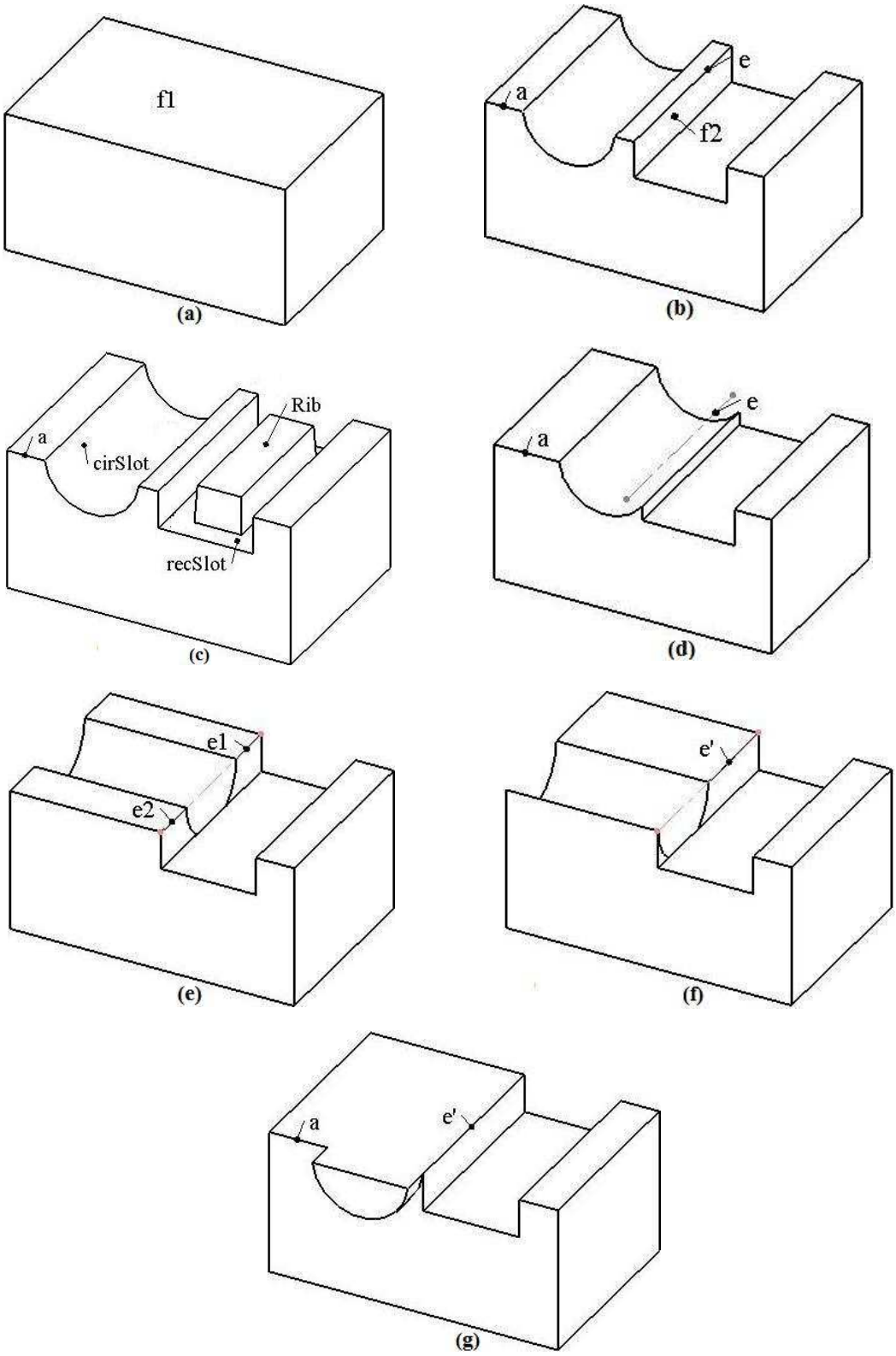


Fig. 4.8 Topological edges

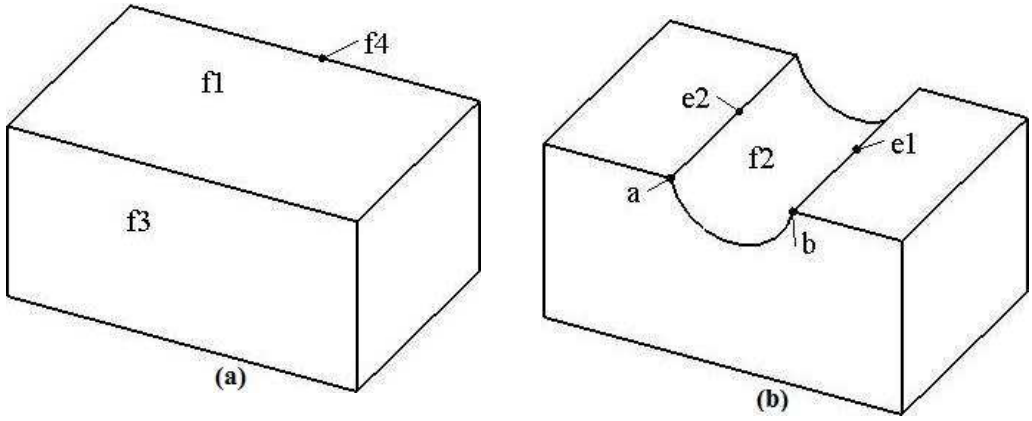


Fig. 4.9 Edge naming

4.3.3 Operation Validity

Although the attached face in an operation can be identified on the modified design model, the validity of the design model needs to be re-evaluated in some cases. As shown in Fig. 4.10(a), a design model has three features, namely, *cirSlot*, *rectSlot* and *Step*. *DS*(*cirSlot*) is *rectSlot*, *cirSlot* is modified at *Site*1, and at the same time a new feature *Rib* is attached to the initial *Stock* at *Site*2, as shown in Fig. 4.10(b, c). However, due to the interaction between the modified *cirSlot* and the new feature *Rib*, the design model becomes invalid when the concurrent operations are synchronized. As shown in Fig. 4.10(d), the top of *cirSlot* is blocked by *Rib*. This is a shortcoming of current feature-based modeling where the validities of the design features cannot be maintained during the design process (Bidarra and Bronsvoort, 2000). In replicated collaborative design, this problem is more critical. Since the two conflicting operations are performed by two designers, they need to discuss with one another to resolve this problem. In the proposed concurrency approach, after the execution of an operation, the designer needs to send a message to the operation initiator. If the two concurrent operations are conflicting and causing the design model to be invalid, a success message of the execution will not be sent; instead, they will resolve this problem collaboratively.

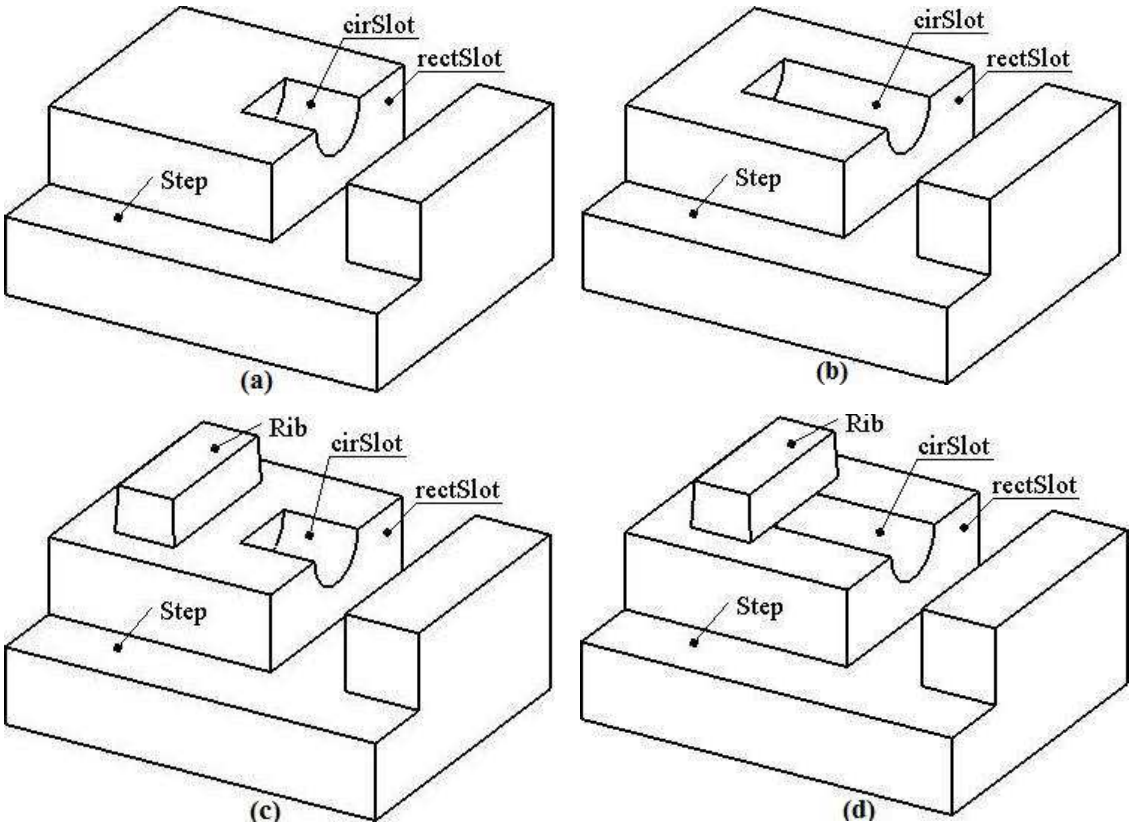


Fig. 4.10 Model validation

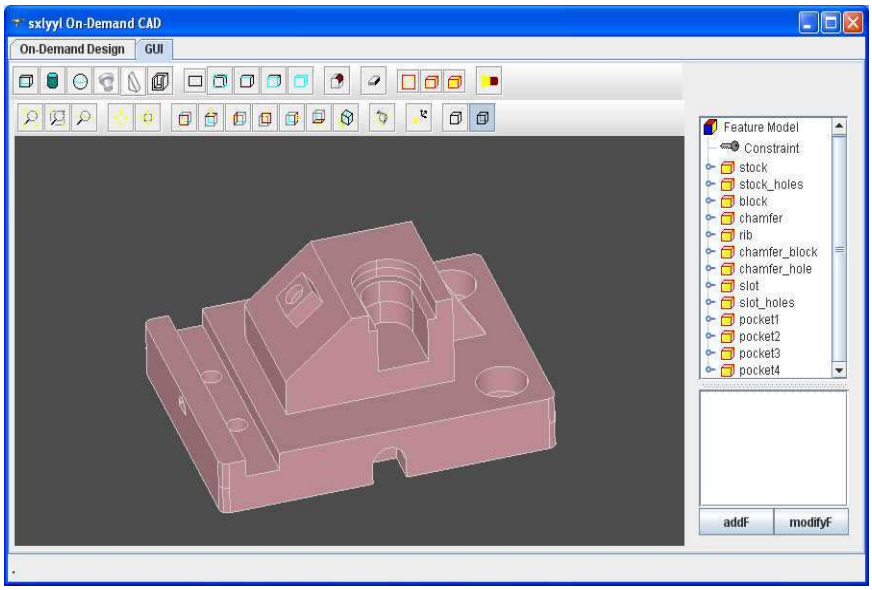
4.4 Case Study

A proof-of-concept prototype system for the proposed concurrency control approach has been developed based on JDK 1.6 and the Open CASCADE. Fig. 4.11 shows the graphical user interface, and a case study with the directed acyclic graph (DAG) of the design features. According to the concurrency control approach, the case study model can be divided into independent portions. At most, four ‘modify feature’ operations and one ‘create feature’ operation can be processed concurrently, as shown in Table 4.1. Since the dependency relationship is basically a parent-child relation, the effectiveness of the concurrency control is restricted by the parent-child relations of the features on a design model. If the design features have more generations, more feature portions can be divided and edited concurrently by using this coordination approach. The extreme case is when all the design features are attached to the initial

stock, as shown in Fig. 4.12. In this case, only one designer is permitted to edit the design model at any time, and this concurrency control approach becomes a total-locking mechanism.

Table 4.1 Parallel operations

Edited feature	$S-holes2$	$B-hole$	rib	$pocket\ 2$	New feature
DS	$S-holes2$, $slot$	$block2$, $B-hole$	$stock$, $block$, rib	$pocket1$, $pocket\ 2$	$pocket4$



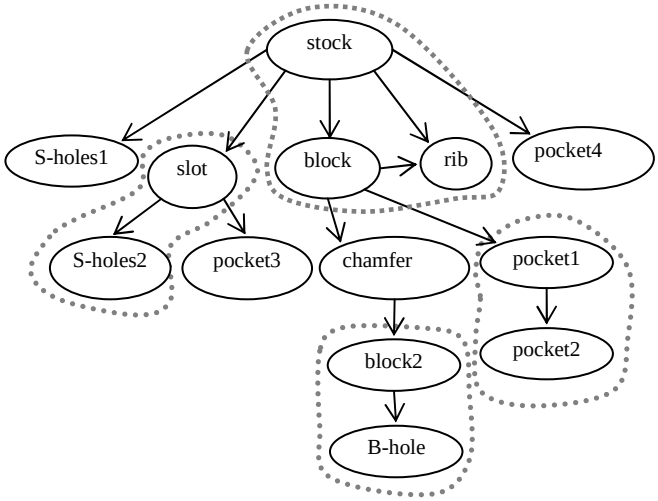
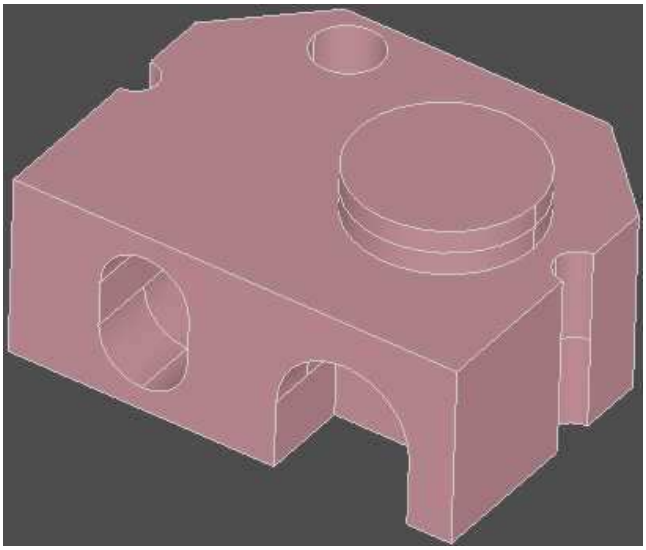


Fig. 4.11 Case model



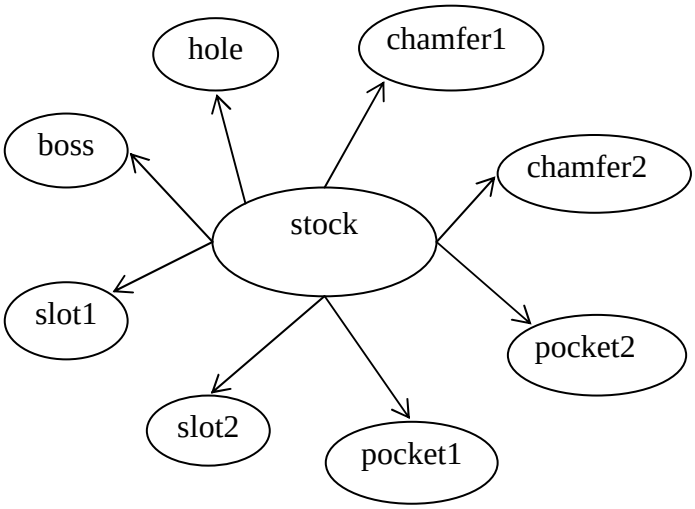


Fig. 4.12 An extreme case

4.5 Summary

In this Chapter, a granular locking mechanism has been proposed as the coordination mechanism for replicated collaborative feature modeling. The dependency scope of a feature, which includes its direct ancestral features, direct descendant features and the feature itself, is employed as the locking granularity. At any time, more than one 'modify operation' can be executed concurrently as long as their dependency scopes are mutually exclusive. However, only one 'create operation' is permitted so as to maintain the consistency of the 'feature creation order'. The potential conflicts of design operations caused by feature interactions are resolved using a naming and

matching mechanism, through which the correspondence of the modified topological entities can be achieved correctly. The modified attached faces are identified using a *FaceIdGraph* and the modified reference edges are identified through their adjacent feature faces. The limitation of the proposed approach is that the concurrency effectiveness depends strongly on the parent-child relations of the features on a design model. If the design features have more generations, more feature portions can be divided. When all the features are attached or constrained to the initial stock, they have only one direct ancestor, so they are within only one dependency scope. Hence, all the features need to be locked together each time, and the proposed coordination approach degrades to be the total-locking mechanism.

Chapter 5 Freeform Feature Modeling

5.1 Introduction

The success of a new product does not only depend on its quality and short development time, but also rely on its attractive and pleasing appearance. Under this consideration, freeform feature modeling has been proposed to facilitate users to create and manipulate freeform surfaces, which are commonly described in Bézier, B-spline and Non-Uniform Rational B-Spline (NURBS) formats. As presented in the review Section 2.1.3, current freeform feature modeling has some problems and weaknesses, which require further research effort. Firstly, the specification of freeform features is not straightforward as in regular feature modeling. Since the essence of freeform feature modeling is to create and represent a design model using freeform solutions, restricting a freeform feature to a geometric shape may contradict this essence. Secondly, since the boundary of freeform models is not planar, the boundary surface of a freeform feature may not contact the attach modify surface seamlessly, and the attachment operation of a freeform feature to a base model becomes a challenge. Thirdly, the smoothness across the boundary curve of two freeform surfaces becomes a crucial issue in freeform feature modeling. In product design, a tangential or even higher smoothness across the boundary curve is usually needed, and this issue has attracted many research studies.

In this Chapter, some issues in freeform feature modeling will be discussed. Specifically, in the first section, the specification of volumetric freeform features is presented and discussed, which may be used for creating simple models in conceptual design; in the second section, a surface blending approach used in displacement feature

modeling is elaborated; in the third section, the displacement feature modeling is fit into a collaborative design environment, in which the coordination and synchronization mechanisms are discussed briefly.

5.2 Specification of Volumetric Freeform Features

5.2.1 3D Constraint Solving

Approaches to constraint solving in 3D space have been reported in the literature, in which a graph-based approach is an effective approach (Du and Hwang, 1995; Durand and Hoffmann, 2000). In the graph-based approach, the constraint problem is constructed as a graph, in which the nodes represent the geometric entities and an edge between two nodes represent a constraint between the two geometric entities. The geometric entities considered are points and planes, and the constraints allowed are distance between two points, distance between a point and a plane, and angle between two planes. A point is represented by its Cartesian coordinates, $p:(p_x, p_y, p_z)$, and a plane is represented by the direction cosines of the unit normal and the signed distance from the origin, $P:(n_x, n_y, n_z, d)$, where $n_x^2 + n_y^2 + n_z^2 = 1$. There are two general phases for the graph-based approach, namely constraint graph analysis and geometry construction. In the first phase, the characteristic of the constraint graph is analyzed and the constraints in the graph are decomposed into clusters of geometric entities that are placed with respect to one another. In the second phase, all the clusters are combined using a recursive technique, resulting in a valid placement for all the geometric entities. Placing a new entity requires that it is constrained by the three already known entities, since each geometric entity has three degrees of freeform. Hence, in the construction of a cluster, a set of three pairwise constrained nodes is necessary, and a new entity is added to the cluster if it is incident to three nodes

already in the cluster. However, there may eventually be unused constraints in the remaining initial graph, yet no new cluster can be started. In this case, any remaining constraint and its two incident nodes would form a degenerate cluster.

The above construction process is illustrated in Fig. 5.1, where the problem is to place the six vertices of the 3D object shown on the left. The lengths of the edges between the vertices are the constraints. In the first analysis phase, the first cluster is constructed using nodes r, s, and t, and its edges are labeled 1. The second cluster is constructed using nodes u, v, and w, and its edges are labeled 2. The remaining constraints cannot be added to the constructed clusters, so they generate degenerate clusters, labeled 3, 4, and 5. In the second construction phase, all the clusters are merged. This constraint problem is under-constrained, so it needs another three distance constraints, which are (t,u), (t,v), and (s,u) to make the configuration rigid, resulting in a octahedron.

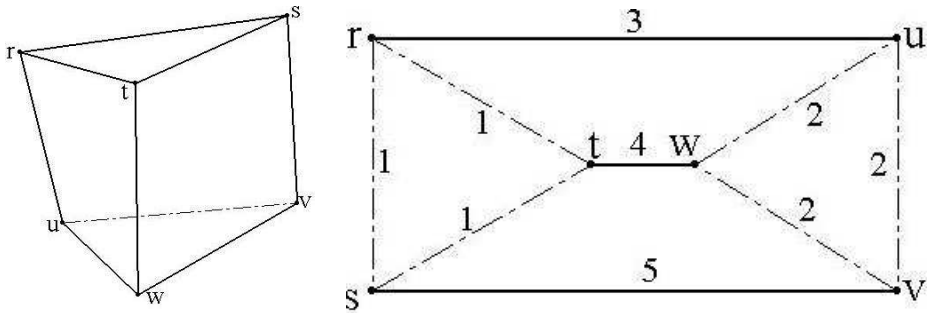


Fig. 5.1 A three-dimensional object and its constraint graph (Du and Hwang, 1995)

5.2.2 Geometric Constraint in Volumetric Freeform Features

In the geometry description of a regular feature, a cross-section is usually swept along a trajectory to create the feature shape. The cross-section is a 2D sketch comprising of geometric entities and constraints. The geometric entities can be points and lines, and the constraints allowed are distance between two points, distance between a point and

a line, and angle between two lines. As shown in Fig. 5.2, the geometric entities, four points p_1, p_2, p_3, p_4 and four lines l_1, l_2, l_3, l_4, can be configured by certain constraints, namely, d_1, d_2, d_3, d_4, d_5 and angle constraints a, b, shown on the left. When the geometric entities are placed and configured, the generated 2D sketch can be swept along a path (in red color) to create a feature shape, as shown on the right in Fig. 5.2.

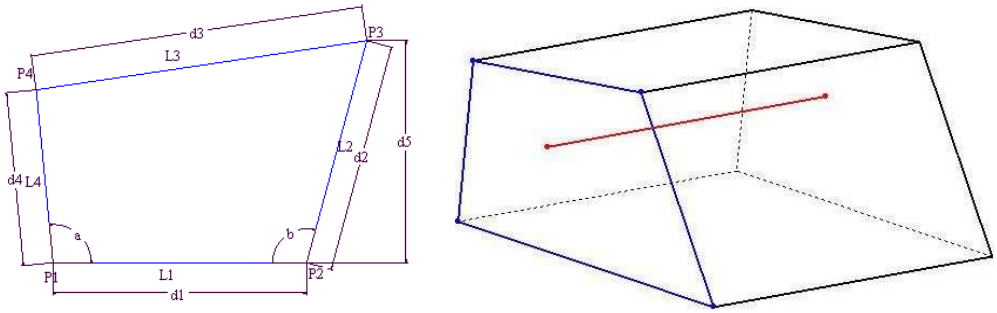
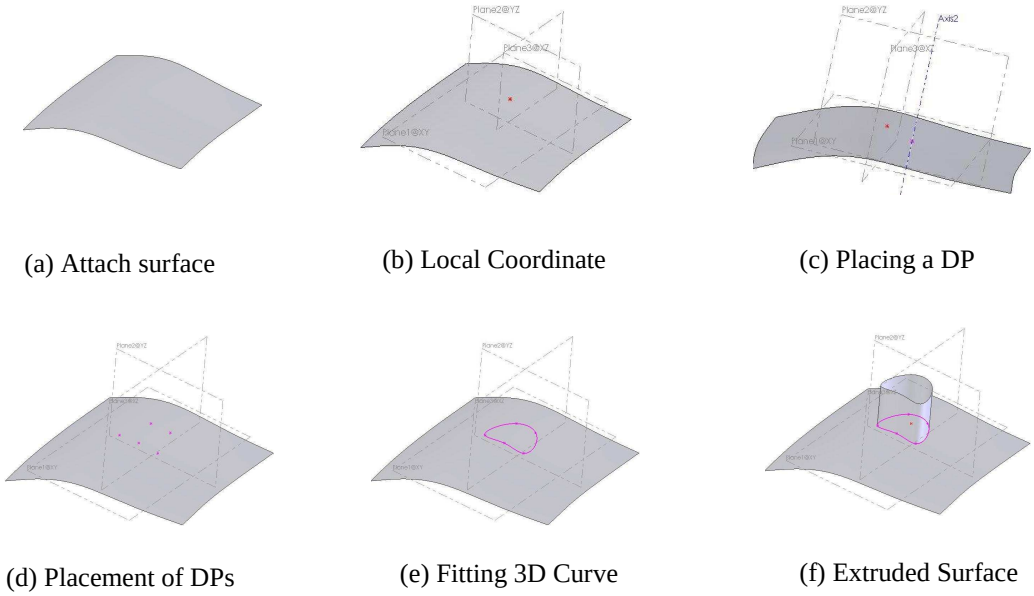


Fig. 5.2 2D sketch and the swept shape

Likewise, the shapes of volumetric freeform features can also be described with the sweeping operation. In the geometry description, an enclosed 3D profile curve is swept along a 3D trajectory curve, and the two ends of the swept surface are enclosed by the two cap faces. The 3D curves used are generated by interpolating some definition points (DPs), which are points in 3D space represented by their Cartesian coordinates, $p:(p_x, p_y, p_z)$. The difference from regular features is that the 3D curve needs to be fitted to the attach surface in the model seamlessly, thus the feature attachment operation can be performed correctly. The main task is to place the DPs and generate the interpolating 3D curve on the attach surface seamlessly, after which the feature shape can be generated by some standard sweeping operations.

Since each geometric point has three degrees of freeform, a new DP can be positioned uniquely in 3D space by constraining it to three already known geometric entities. In the placement operations of DPs, the generic positions of the three initial entities

depend on the geometric information determined by user input. In this work, there are four initial geometric entities, namely one point and three plane, which constitute the local coordinate system for the feature shape. The initial point (IP) is the local origin, and the three initial planes (IPs) are the local coordinate planes, XY, YZ, and XZ plane. At the beginning of feature instantiation, the IP is placed on the attach surface and the z coordinate axis is parallel to the normal vector of the attach surface at the IP point. Through this initial placement, the IP and the IPs can be positioned using the geometric information determined by user input. After that, the DPs can be positioned by solving the constraints between themselves and the initial entities. Since the DPs of the 3D curve need to be placed on the attach surface, only two distance constraints are defined between the DPs and the local XZ, YZ plane, and the other degree of freeform is determined by the attach surface. The constraint specification is as follows: firstly, since the distance between a point $p_1 : (p_x, p_y, p_z)$ and a plane $P_2 : (n_x, n_y, n_z, d_2)$ is $d_{12} : p_1 \bullet n_2 = d_2 + d_{12}$, the distance between a DP p and the local XZ plane is $d_1 : p_y = d_1$, and the distance between a DP p and the local YZ plane is $d_2 : p_x = d_2$; secondly, since the two distance constraints of a DP can determine two planes, $p_y = d_1$ and $p_x = d_2$, the DP will locate on the intersection line l of the two planes. In this case, the DP is positioned at the intersection point of the line l and the attach surface. Through the above constraint specification, all the DPs are configured and placed on the attach surface seamlessly. The procedure of the placement operations of DPs is illustrated in Fig. 5.3, where an extruded feature face is created.



(a) Attach surface (b) Local Coordinate (c) Placing a DP

(d) Placement of DPs (e) Fitting 3D Curve (f) Extruded Surface

Fig. 5.3 Placement of definition points

5.2.3 3D Profile Curve Generation

As presented in previous section, the 3D curve of a sweeping freeform feature can be generated by interpolating the DPs that are placed on the attach surface. However, the interpolating curve may be not lying completely on the attach surface if standard interpolation operations are performed directly on the DPs, thus the boundary of the swept freeform surface is not seamlessly matched with the attach surface. The desirable 3D curve should be a B-spline curve in control points representation and lying completely on the attach surface, which can be used as a trimming curve. The trimming curve on a freeform surface is usually first computed in the parametric domain of the surface, and then represented in space form as the mapping of the domain curve on the surface (Renner and Weiβ, 2004; Yang *et al*., 2008). In this research, the parametric values of a DP can be computed by using inverse parameterization techniques, and a corresponding point in the parametric domain is found, termed Domain Definition Point (DDP). Hence, all the corresponding DDPs of the DPs can be computed, and a domain curve is generated by interpolating the DDPs,

termed domain sketch. The domain sketch is then mapped to the attach surface as the boundary curve of a freeform feature. In such an approach, it ensures that the boundary lies on the attach surface seamlessly, thus the contact between the swept surface and the attach surface has no gap at all.

In Summary, a volumetric freeform feature can be defined using a cross-section 3D curve and a trajectory 3D curve, which are obtained by interpolating certain 3D points. Once the 3D points are positioned in 3D space using constraint solving, a volumetric freeform surface can be generated using standard sweeping techniques. Since the cross-section needs to lie on the attach surface seamlessly, a parametric curve can be first interpolated in the domain space, which is then evaluated in the attach surface. In this way, the generated freeform shape is attached to the base surface seamlessly. However, the swept freeform shapes can only be used in certain conceptual designs for initial review of the product model. It is due to the fact that the feature shape here is restricted by certain 3D points and the standard sweeping operations, and is due to the fact that the generated freeform surface is not described using the commonly used representations, e.g., Bézier, B-spline, NURBS.

5.3 Displacement Feature Modeling

Displacement feature is a type of freeform surface features that is commonly used in industrial parts. As presented in the review Section 2.1.3.3, there are three major steps in displacement feature modeling: firstly, the boundary curve is specified on the base surface; secondly, the modified surface region is trimmed and displaced towards the exterior or interior of the base surface; thirdly, the transition surface is generated by a surface blending approach. The two crucial issues in the modeling procedure are

discussed in this section, namely, the specification of the boundary curve and the surface blending approach.

5.3.1 Boundary Curve Specification

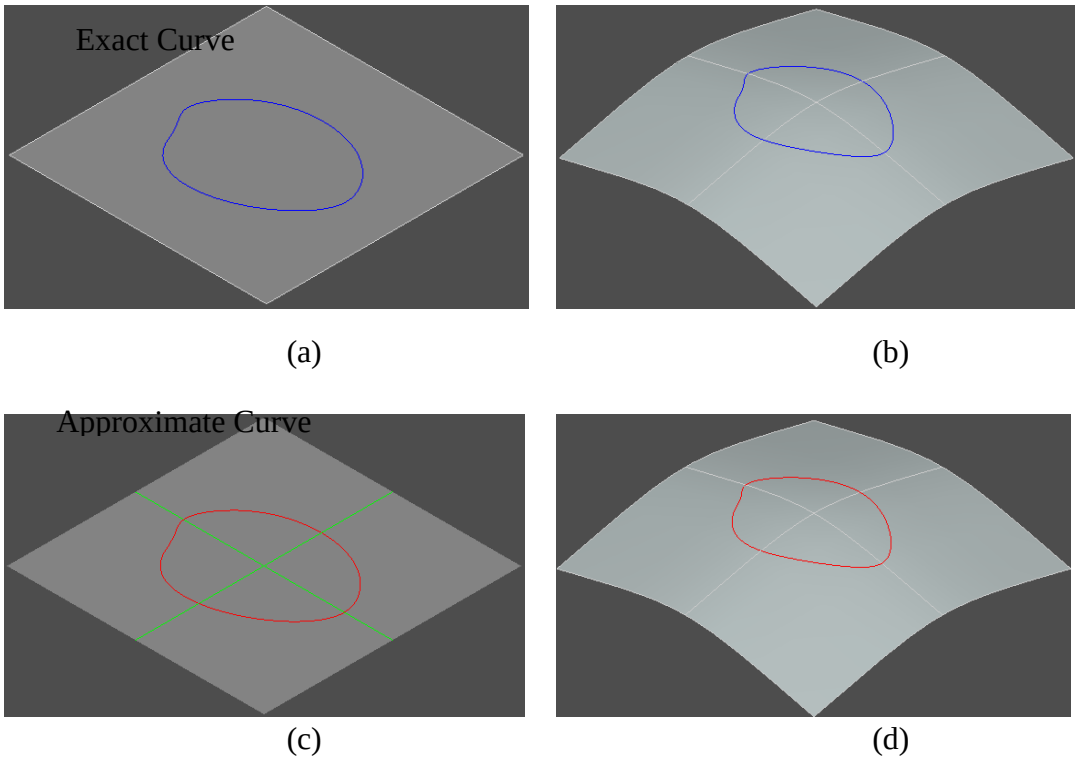


Fig. 5.4 Exact curve and approximated curve

In this work, the boundary curve in a displacement feature is an exact curve lying on the base surface to ensure that the continuity across the boundary is at least G^0. Alternatively, the approximation algorithm reported by Yang *et al*. (2008) may be used to generate a boundary curve lying on the base surface seamlessly. Firstly, the domain curve of the boundary curve is approximated by a polyline, and the base surface is divided into Bézier surface patches. Secondly, the approximated polyline is projected to the Bezier surfaces to generate certain Bézier curves. In this approximation, the Hausdorff diatance between the approximated 3D curve and the exact 3D curve, and the tangent discrepancy between any pair of adjacent 3D Bézier curves are both under the user-specified tolerance. As in Fig. 5.4, (a) shows a B-spline domain curve; (b)

shows the mapped 3D curve of the exact domain curve computed using the modeling algorithms in Open CASCADE; (c) shows the approximated polyline of the domain curve; (d) shows the mapped 3D curve of the polyline. Table 5.1 gives the comparison between the mapped 3D curve of the exact domain curve and that of the polyline.

Table 5.1 Comparison between exact curve and approximated curve

	Tolerance	Degrees	Number of control points	Number of segments	Continuity
Exact curve	-	7	50	8	G^1
Approximated	1.0×10^{-2}/5°	3	376	125	5°-G^1

The shortcomings of this approximation algorithm are as follows. Firstly, although the mapped space curve lies completely on the base surface, its degree, $(d_u + d_v)$ where d_u and d_v are the surface degrees, cannot be decreased as specified by users. Secondly, there are too many segments of the mapped space curve, which may make the surface blending complicated if it is used in this work. Thirdly, the adjacent Bézier curves on the base surface have normal discrepancy, which will remain in the blending surface. In addition to the shortcomings of this algorithm itself, the mapping algorithms provided in OCC is basically an approximation technique, which samples the domain curve and interpolates the sample points using a continuous B-spline curve. That is why the degrees of the exact mapped curves and the approximated 3D curve in Fig. 5.4 are not (3+3)*3 and (3+3)*1, but 7 and 3 respectively.

In this work, the boundary curve is generated using Maple, which generates an exact 3D curve lying on the base surface. The generated 3D curve using Maple is not represented in Bézier representation, but in the power basis representation, which can be converted to Bézier using an available algorithm, which will be elaborated in Chapter 6.

5.3.2 The Proposed Surface Blending Approach

5.3.2.1 Algorithm Overview. A Bézier curve is defined by

$$\mathbf{C}(t) = \sum_{i=0}^{p} B_{i,p}(t)\mathbf{P}_i$$

where $\mathbf{P}_i$ are the control points, and $B_{i,p}$ are the p th-degree Bézier basis functions.

A Bézier surface in the 3D space is defined by

$$\mathbf{S}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_{i,n}(u)B_{j,m}(v)\mathbf{P}_{i,j}$$

where $\mathbf{P}_{i,j}$ are the control points, and $B_{i,n}(u)$ and $B_{j,m}(v)$ are the n th-degree and m th-degree Bézier basis functions, respectively.

The Cubic Hermite Interpolant is denoted in Eq. (5.1),

$$\mathbf{S}(t,v) = h_{00}(v)\mathbf{C}_1(t) + h_{01}(v)\mathbf{C}_2(t) + h_{10}(v)\mathbf{T}_1(t) + h_{11}(v)\mathbf{T}_2(t) \qquad (5.1)$$

where $\mathbf{C}_1$ and $\mathbf{C}_2$ are the boundary curves and $\mathbf{T}_1$ and $\mathbf{T}_2$ are the tangent fields along the boundary curves.

In this work, the Cubic Hermite Interpolant is approximated for surface blending, so the blending surface is a $n\times 3$ Bézier surface $\mathbf{S}(t,v)$, where n is the degree of the boundary curve $\mathbf{C}_1(t)$. In order to ensure that the blending surface is tangential to the base surface, the connection between the t isocurve of the blending surface and the base surface can be interrogated. If the t isocurve is tangential to the base surface, it assures that the blending surface has the tangential smoothness with the base surface.

The t isocurve of the blending surface can be obtained as follows:

For fixed $t = t_0$,

$$\mathbf{C}_{\mathbf{t_0}}(v) = \mathbf{S}(t_0, v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_{i,n}(t_0)B_{j,m}(v)\mathbf{P}_{i,j} = \sum_{j=0}^{m} B_{j,m}(v)(\sum_{i=0}^{n} B_{i,n}(t_0)\mathbf{P}_{i,j}) = \sum_{j=0}^{m} B_{j,m}(v)\mathbf{Q}_j(t_0) \quad (5.2)$$

where $\mathbf{Q}_j(t_0) = \sum_{i=0}^{n} B_{i,n}(t_0)\mathbf{P}_{i,j}$, is a Bézier curve lying on the blending surface.

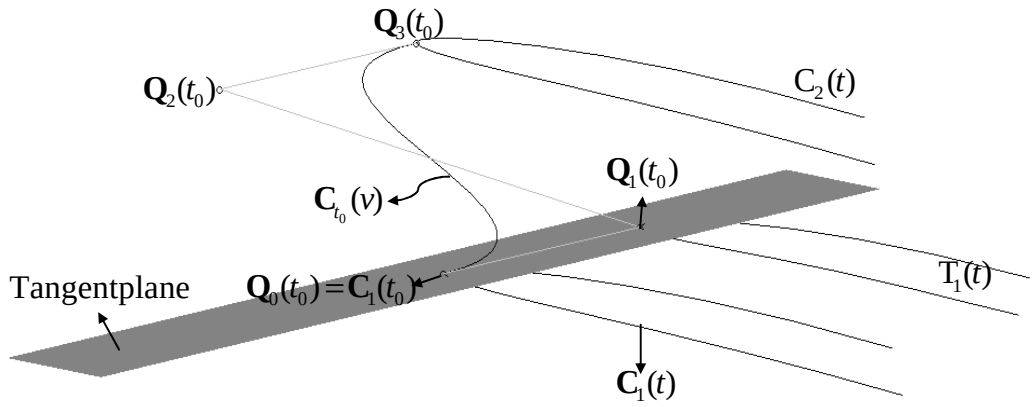


Fig. 5.5 t isocurve and the relevant curves

This indicates that the t isocurve of the blending surface is a Bézier curve $\mathbf{C}_{\mathbf{t_0}}(v)$, in which the control points are $\mathbf{Q}_j(t_0)$ ($j = 0...3$), and $\mathbf{C}_{\mathbf{t_0}}(v)$ is attached to the base surface at point $\mathbf{C}_1(t_0)$, as shown in Fig. 5.5. If the tangent vector of the t isocurve $\mathbf{C}_{\mathbf{t_0}}(v)$ at point $\mathbf{C}_1(t_0)$ is on the tangent plane of the base surface at point $\mathbf{C}_1(t_0)$, $\mathbf{C}_{\mathbf{t_0}}(v)$ is tangential to the base surface at point $\mathbf{C}_1(t_0)$. Hence, one can now interrogate the tangent vector of the t isocurve $\mathbf{C}_{\mathbf{t_0}}(v)$ at point $\mathbf{C}_1(t_0)$, which is given in Eq. (5.3), where $m = 3$ and $\mathbf{Q}_0(t_0)$, $\mathbf{Q}_1(t_0)$ are the first two control points of $\mathbf{C}_{\mathbf{t_0}}(v)$.

$$\mathbf{C}'_{\mathbf{t_0}}(0) = m(\mathbf{Q}_1(t_0) - \mathbf{Q}_0(t_0)),\ \mathbf{C}'_{\mathbf{t_0}}(1) = m(\mathbf{Q}_m(t_0) - \mathbf{Q}_{m-1}(t_0)) \quad (5.3)$$

From Eq. (5.2), it can be obtained that

$$\mathbf{Q}_0(t_0) = \sum_{i=0}^{n} B_{i,n}(t_0)\mathbf{P}_{i,0},\ \mathbf{Q}_1(t_0) = \sum_{i=0}^{n} B_{i,n}(t_0)\mathbf{P}_{i,1},$$

where $\mathbf{P}_{i,0}$ and $\mathbf{P}_{i,1}$ are the first two column control points of the blending surface. $\mathbf{P}_{i,0}$ is also the control point of the boundary curve $\mathbf{C}_1(t)$, which ensures that the blending surface is attached to the base surface seamlessly along the boundary curve.

From above, in order to achieve the tangential connection between $\mathbf{C}_{\mathbf{t_0}}(v)$ and the base surface, the control point $\mathbf{Q}_1(t_0)$ needs to be assigned on the tangent plane of the base surface at point $\mathbf{C}_1(t_0)$. Stated differently, for fixed $t = t_0$, the point on the Bézier curve $\mathbf{Q}_1(t) = \sum_{i=0}^{n} B_{i,n}(t)\mathbf{P}_{i,1}$ should be on the corresponding tangent plane. The cause-effect relation in the proposed algorithm is illustrated in Fig. 5.6.

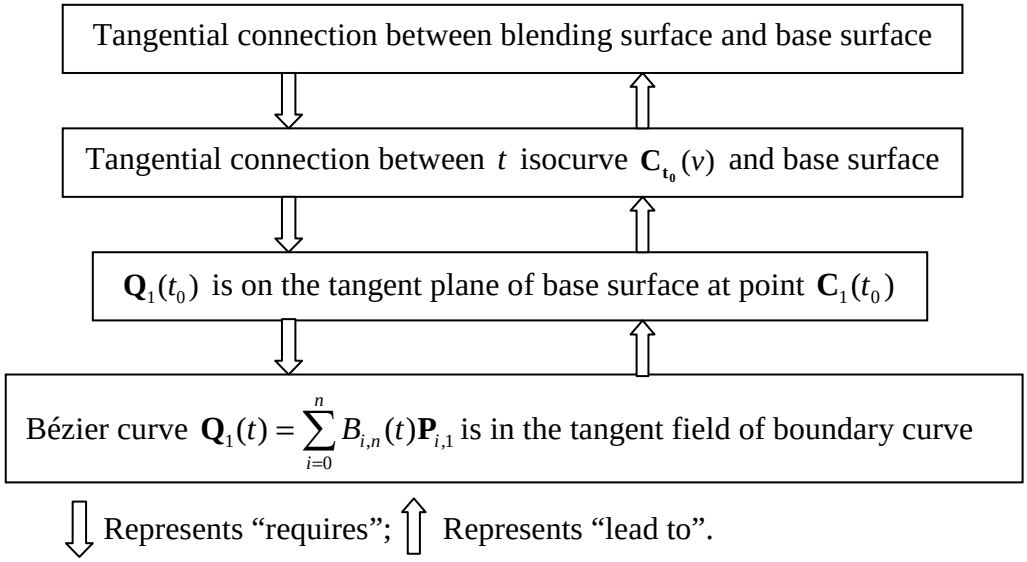


Fig. 5.6 Cause-effect relation in the proposed algorithm

Consequently, a Bézier curve $\mathbf{Q}_1(t)$ that is in the tangent field of the boundary curve needs to be obtained. To achieve this goal, the main task in this research is to offset the boundary curve on the tangent planes for a certain distance to obtain the control points $\mathbf{P}_{i,1}$, which are used as the control points of the blending surface. The offset curve,

which is the Bézier curve $\mathbf{Q}_1(t) = \sum_{i=0}^{n} B_{i,n}(t)\mathbf{P}_{i,1}$, ensures that the point $\mathbf{Q}_1(t_0)$ is on the corresponding tangent plane, and hence the t isocurve $\mathbf{C}_{\mathbf{t_0}}(v)$ is tangential to the base surface. For offsetting the boundary curve, a number of points is first sampled and offset on the corresponding tangent planes. Next, these offset sample points are interpolated as a B-spline curve. The main algorithm flow is described as follows.

1. Sample points on the boundary curve and determine the offset direction vectors at these sample points.
2. Translate the sample points along the offset vectors obtained and interpolate the offset points as a B-spline curve.
3. Knots refine the boundary curve and the offset curve for surface blending.

For constructing the entire blending surface, the modified surface region is displaced towards the interior or exterior of the base surface. Hence, the boundary curve $\mathbf{C}_1(t)$ is also displaced, which forms the other boundary curve $\mathbf{C}_2(t)$ of the blending surface. Analogous to the offset of $\mathbf{C}_1(t)$, $\mathbf{C}_2(t)$ is offset on the tangent planes of the displaced surface region to obtain a Bézier curve $\mathbf{Q}_2(t) = \sum_{i=0}^{n} B_{i,n}(t)\mathbf{P}_{i,2}$, which ensures that the point $\mathbf{Q}_2(t_0)$ is on the corresponding tangent plane. Finally, all the four column control points of the blending surface $\mathbf{S}(t,v)$ are obtained to generate the transition geometry.

5.3.2.2 Surface Blending. Points sampling is based on the bounds on the second derivatives. For parametric curves, the number of sample points for equally spaced parameters on [0, 1] is computed as follows (Piegl and Tiller, 1999):

$$n = (\frac{1}{\varepsilon})^{pow}\sqrt{\frac{M}{8}}$$

where $pow = \begin{cases} 0.5 & \text{linear approximation} \\ 0.34 & \text{otherwise} \end{cases}$. M is the bound on the second derivative of the offset curve and ε is a user-defined tolerance.

For a Bézier curve, $2(p+1)$ can be sampled for computing the second derivatives at these points, where p is the degree of the Bézier curve, and the magnitude of the maximum of these derivatives is used for M (Piegl and Tiller, 1999). A default number $n = p+1$ is introduced by Piegl and Tiller (1999) to ensure that a small curve segment can be sampled properly when the tolerance is large. The offset direction vector at the sample point is determined by

$$\mathbf{T}(t) = \pm(\frac{d\mathbf{C}(t)}{dt} \times \mathbf{n}(u(t), v(t))),$$

where $\frac{d\mathbf{C}(t)}{dt}$ is the tangent vector of boundary curve $\mathbf{C}(t)$, and $\mathbf{n}(u(t), v(t))$ is the normal vector in the surface for each point along $\mathbf{C}(t)$. For each sample point $\mathbf{p}_i$ in the boundary curve, the line $\overline{\mathbf{p}_i\mathbf{q}_i}$ is parallel to the respective offset vector, where $\mathbf{q}_i$ is the offset sample point. Hence, the parameter values at $\mathbf{q}_i$ should correspond to the parameters at $\mathbf{p}_i$, which are set as

$$\bar{u}_i = \frac{i}{n-1}(0 \le i < n),$$

where n is the sampling number. Once the parameterization of the offset sample points has been set properly, a number of interpolation schemes can be used to interpolate the offset sample points to a B-spline curve (Farouki and Sverrisson, 1996). In this work, one can simply use the interpolation functions provided in Open

CASCADE (2009). This algorithm interpolates a B-spline curve passing through an array of points, where the parameters of each of the points can be given.

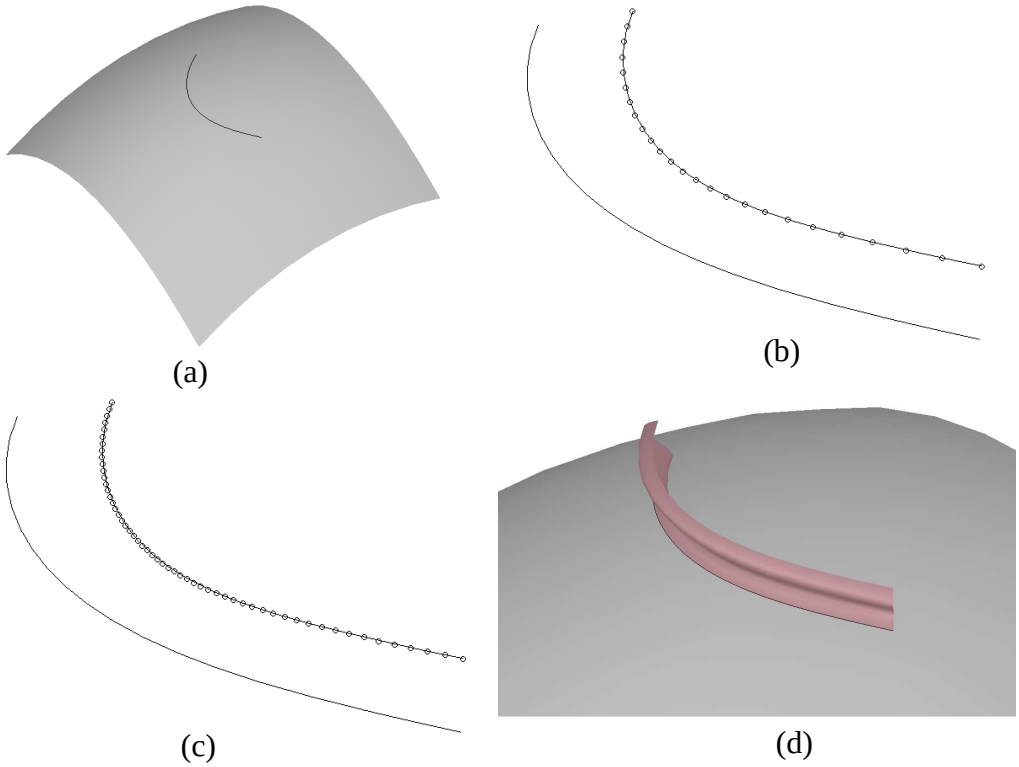


Fig. 5.7 Example#1 of offset curve and blending surface: (a) curve lying on surface; (b) offset curve with $\varepsilon = 10^{-4}$; (c) offset curve with $\varepsilon = 10^{-5}$; (d) blending surface with $\varepsilon = 10^{-5}$.

As shown in Fig. 5.7(a) a quadratic Bézier parameter curve is evaluated in a cubic and quadratic Bézier surface, resulting in a 10th-degree Bézier curve lying on the surface. In Fig. 5.7(b-c), the offset curves with two different tolerances are shown. Since the boundary curve is simple, the interpolating curve of the offset points is basically a Bézier curve. Hence, the control points of the boundary curve and that of the offset curve are used for interpolating the blending surface, as shown in Fig. 5.7(d).

In Fig. 5.8, a 4th-degree Bézier curve is evaluated in a cubic and quadratic Bézier surface, resulting in a 20th-degree Bézier curve lying on the surface. In Fig. 5.8(b-c), the offset curves with two different tolerances are shown. Since the boundary curve is

more complex, the interpolating curve of the offset points is a B-spline curve with three interior knots. Analogously, the offset curve of the displaced boundary curve is a B-spline curve with one interior knot. To generate the blending surface, the boundary curves and the offset curves are converted to the same degree and in the common knot vector. Finally, the compatible B-spline curve has four interior knots, and the B-spline curves are converted into five Bézier curve segments. The control points of the Bézier segments are used to generate the blending surface, as shown in Fig. 5.8(d), which contains five constitutive Bézier surface patches.

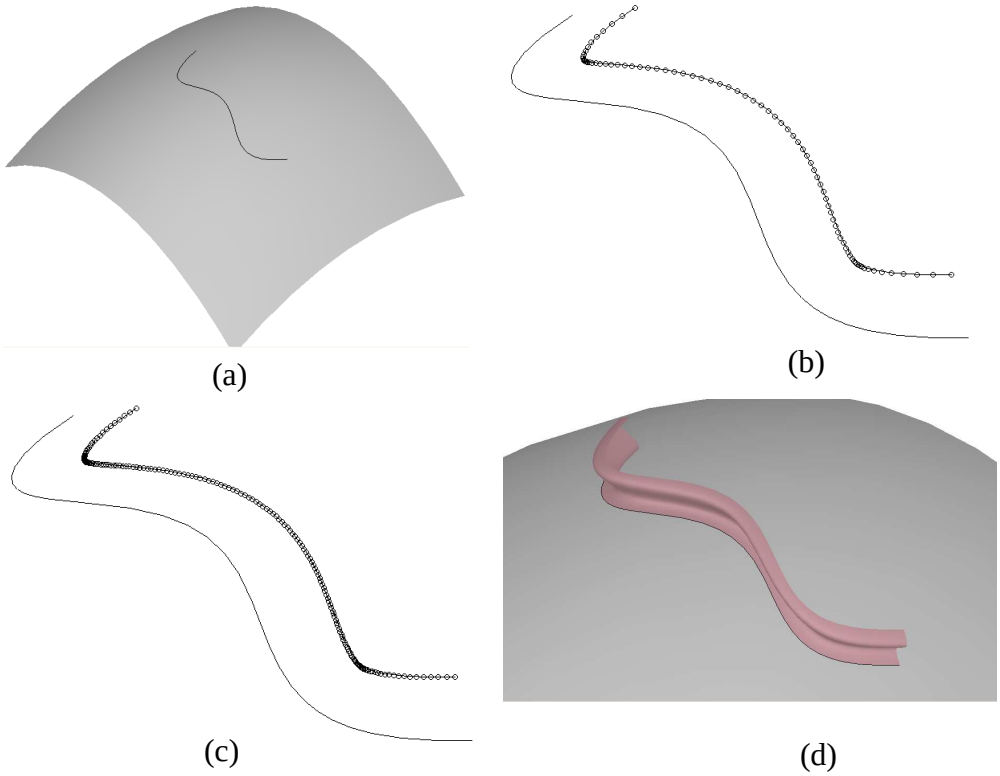


Fig. 5.8 Example#2 of offset curve and blending surface: (a) curve lying on surface; (b) offset curve with $\varepsilon = 10^{-4}$; (c) offset curve with $\varepsilon = 10^{-5}$; (d) blending surface with $\varepsilon = 10^{-4}$.

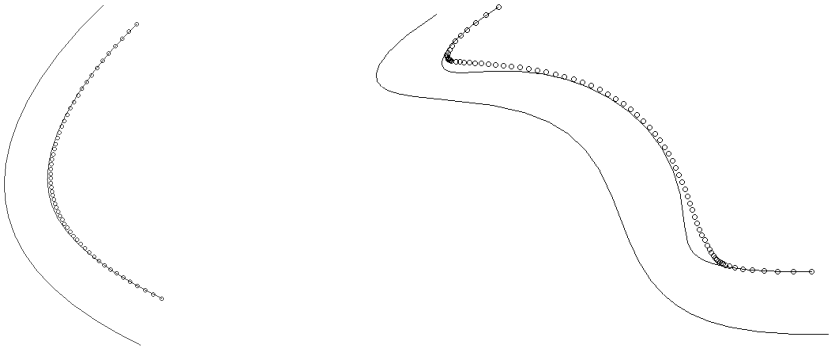
5.3.2.3 Comparison with other works

In Section 5.3.2.3, it is known that the points on the Bézier curve $\mathbf{Q}_1(t) = \sum_{i=0}^{n} B_{i,n}(t)\mathbf{P}_{i,1}$, not the control points $\mathbf{P}_{i,1}$, should be on the corresponding tangent planes. However, in

the work reported by van Elsas's method (1998), the control points of the boundary curve are translated along the vectors on the tangent planes, and it does not guarantee that the offset curve $\mathbf{Q}_1(t)$ is on the tangent planes. It is true that Bézier curves are invariant under the usual transformations, such as rotations, translations, and scaling, which means one applies the transformation to the curve by applying it to the control polygon (Piegl and Tiller, 1997). However, for this property of affine invariance, the transformation vectors must be the same for the entire control polygon.

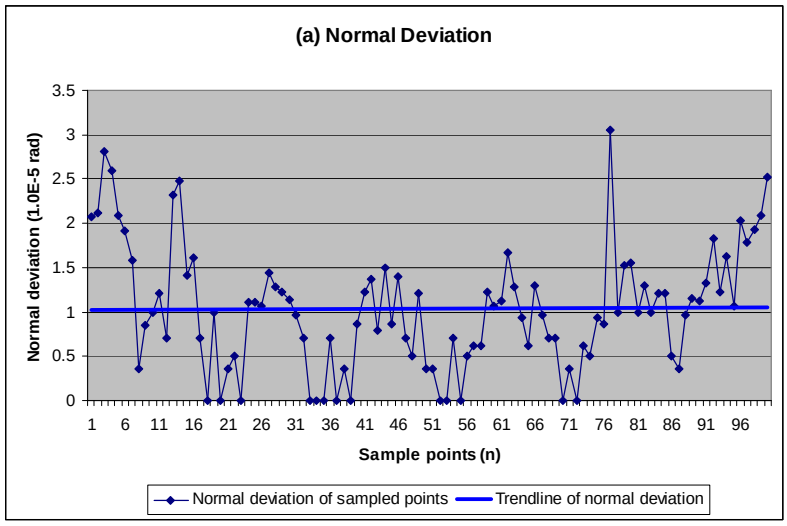
In the work of van Elsas, the translation vectors of each control point of the boundary curve are different, and hence translating the control points along the tangent planes cannot guarantee that the corresponding Bézier curve is on the tangent planes. As a result, the work of van Elsas has the weakness that it may not be able to find the tangent field curve of the boundary curve by translating its control points. However, his approach can be used when accuracy is not strictly needed, such as for certain conceptual illustrations. As shown in Fig. 5.9(a), the sample points are those used in Fig. 5.7(c), which are on the tangent planes, and the Bézier curve is generated by offsetting the control points of the boundary curve. It can be seen that the curve obtained using van Elsas's method interpolates almost all the sample points. However, it is not a reliable approach when the boundary curve and the base surface become more complex. As shown in Fig. 5.9(b), the Bézier curve does not interpolate all the sample points used in Fig. 5.8(b), and has a discrepancy, which means the tangential smoothness has not been achieved here. It is difficult to identify the difference between this study and the work of van Elsas from the visualization of the blending surface viewpoint, but it can be manifested by the normal deviation along the boundary curve, as shown in Fig. 5.10. The normal vectors along the boundary curve are computed on

the base surface and on the blending surface respectively, which are compared to obtain the normal deviation. In the model in this case study, the normal deviation using the proposed method is much smaller, and the difference to that using van Elsas's method is as large as three orders of magnitudes. It should be noted that the offset tolerance here is $\varepsilon = 10^{-4}$. Consequently, for certain conceptual design cases, the boundary curve can be offset simply by offsetting its control points, as in van Elsas's method. However, for a complex boundary curve and surfaces which require higher accuracy, which is quite common in practice, the proposed method is much better for achieving tangential smoothness.



(a) (b)

Fig. 5.9 The offset boundary curve using Elsas's method (1998) does not interpolate the sample points on the tangent planes: (a) example#1; (b) example#2



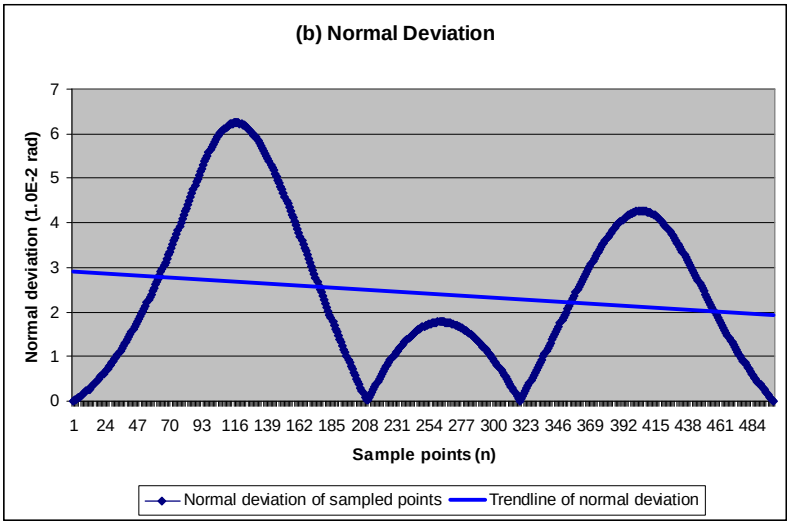


Fig. 5.10 Normal deviation across the boundary curve in Example#2 (a) using the proposed method with $\varepsilon = 10^{-4}$; and (b) using van Elsas's method

In addition, in surface approximation, there is a general trade-off between accuracy and the compactness of the resulting surface. In the work of van Elsas, the control points of the boundary curve are offset directly, and hence the offset curve is compatible with the boundary curve. As the examples in Fig. 5.7 and Fig. 5.8, the boundary curve is a Bézier curve, and the offset curve is also a Bézier curve with the same degree, In this case, the resulting blending surface only has one Bézier surface patch. However, in the present work, the offset curve is generated by interpolating the sampled points, which is usually a B-spline curve having more than one interior knot. When the interpolated B-spline curve is converted into Bézier curves, the B-spline curve is split into more than one curve segments. In order for surface blending, the boundary curve needs to be transformed to be compatible with the offset curve, and it is also split into the same number of curve segments by the interior knots. As the example in Fig. 5.8, when the offset curve and the boundary curve are converted into the same degree and in the common knot vector, the corresponding curves are split into five curve segments, and hence the resulting blending surface has five Bézier patches. As a result, the proposed method can generate more accurate tangential smoothness

across the boundary curve, but it usually generates more surface patches than van Elsas's method.

5.3.3 Self-Intersection Issue

5.3.3.1 Eliminate Self-Intersection in Domain Space.

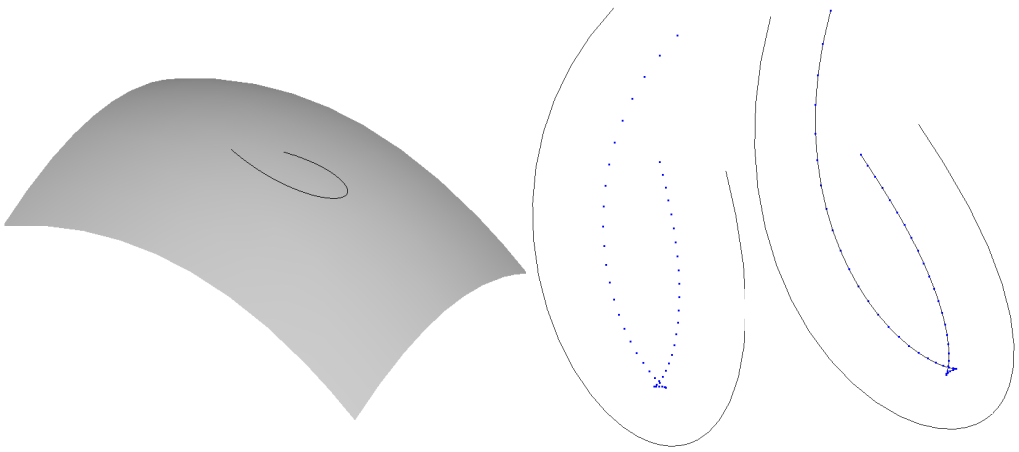


Fig. 5.11 Local self-intersection: (a) curve lying on surface; (b) local self-intersection

Since the boundary curve is offset in the tangent field, the self-intersection issue needs to be addressed. As shown in Fig. 5.11, a 4th-degree Bézier curve is evaluated in a cubic and quadratic Bézier surface, resulting in a 20th-degree Bézier curve lying on the surface, and self-intersection arises when the sample points are offset on the tangent planes. Self-intersection in offset curves and surfaces is a popular research topic, and some related studies have been reported (Pekerman *et al*., 2008; Seong *et al*., 2006). There are two types of self-intersection in the offset curves, namely local and global self-intersection. When the offset distance is larger than the local curvature radius in the original curve, local self-intersection arises. Global self-intersection occurs when two different points in the curve are offset to the same location. In this study, only the local self-intersection will be considered and addressed, as illustrated in Fig. 5.11. Current research in offset curves and self-intersection detection is mainly focused on

planar curves, but this research addresses offset curves in the 3D space. Since the offset of planar curves has been quite well addressed, in this research, self-intersection in the offset boundary curve is transformed to the parameter space of the base surface, which is inspired by the work reported by Flöry and Hofer (2008). In the work by Flöry and Hofer, the curve fitting on the manifolds is carried out as the curve fitting in the parameter space of the manifolds.

When a point $\mathbf{p}_i$ in a surface $\mathbf{S}$ is offset on the corresponding tangent plane $\mathbf{T}_{\mathbf{S}_i}$, the offset direction vector is denoted as $\mathbf{V}(\mathbf{p}_i)$. As it is known, the tangent plane $\mathbf{T}_{\mathbf{S}_i}$ is as the union of tangent vectors to surface $\mathbf{S}$ at the point $\mathbf{p}_i$ (Rovenski, 2000). On the tangent plane, each tangent vector to surface $\mathbf{S}$ can be formulated as the linear combination of the two tangents along iso-parametric curves, $\mathbf{S}_u$ and $\mathbf{S}_v$, which are the vectors calculated using the first partial derivative of $\mathbf{S}$ with respect to u and v. Hence, the offset vector $\mathbf{V}(\mathbf{p}_i)$ can be reformulated as $\mathbf{V}(\mathbf{p}_i) = \Delta u * \mathbf{S}_u(\mathbf{p}_i) + \Delta v * \mathbf{S}_v(\mathbf{p}_i)$, as illustrated in Fig. 5.12.

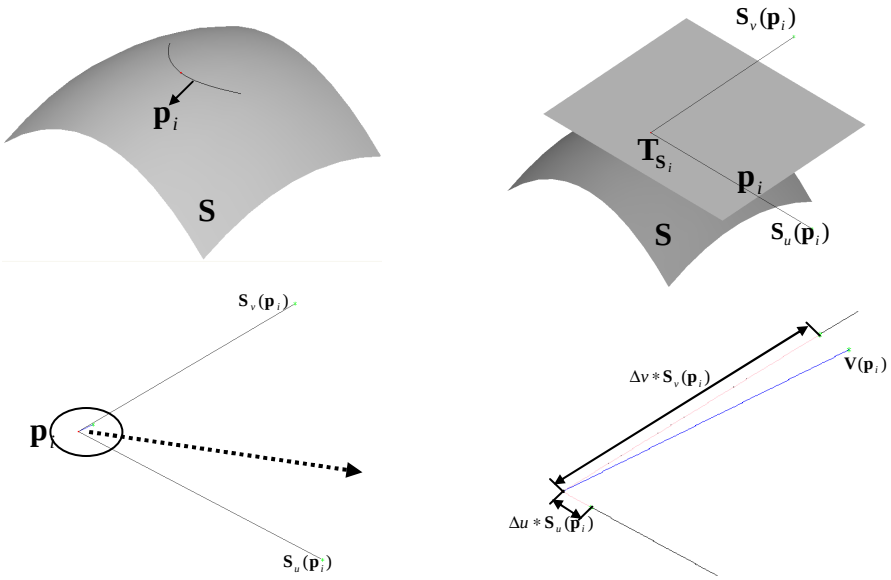


Fig. 5.12 Offset vector and its formulation on tangent plane

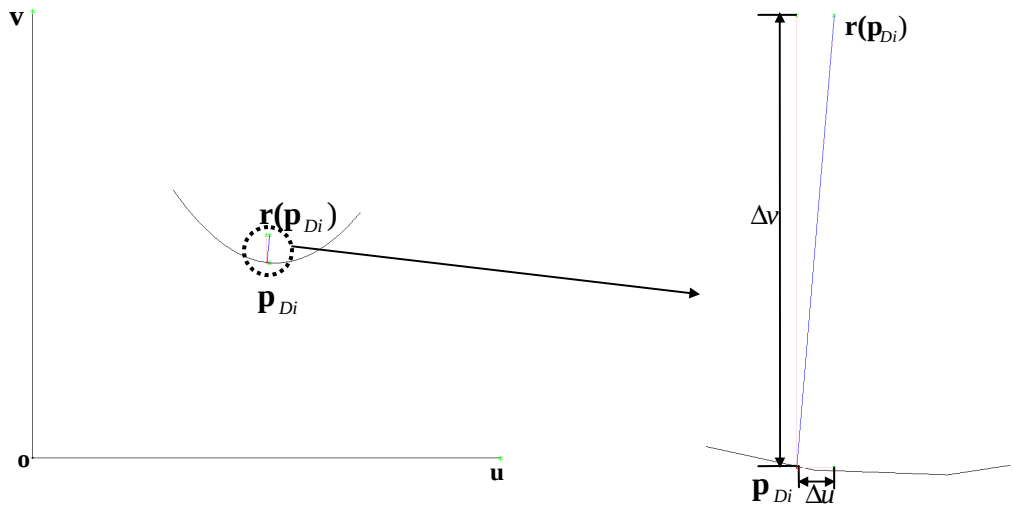
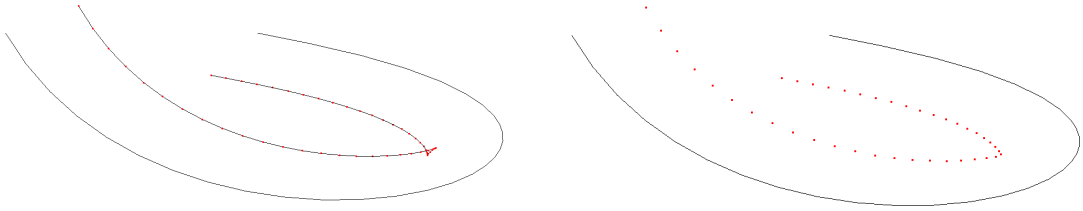


Fig. 5.13 Equivalent offset vector in parameter space

In order to address the self-intersection as in planar curves, one needs to map the offset vectors to a medium plane. Since the offset vector $\mathbf{V}(\mathbf{p}_i)$ implies the change of $\mathbf{p}_i$ on the tangent plane with respect to the changes of the parameters u and v, one can approximately map $\mathbf{V}(\mathbf{p}_i)$ to the changes of parameters in the parameter space of $\mathbf{S}$. Although $\mathbf{p}_i$ is not moved in the base surface, Δu and Δv in the parameter space can represent the move direction of $\mathbf{p}_i$ approximately. Hence, the equivalent offset vector in the parameter space is formulated as $\mathbf{r}(\mathbf{p}_{Di}) = \Delta u\mathbf{u} + \Delta v\mathbf{v}$, as shown in Fig. 5.13, where $\mathbf{u} \times \mathbf{v}$ is the parameter space of $\mathbf{S}$ and $\mathbf{p}_{Di}$ is the parameter point of $\mathbf{p}_i$. Once self-intersection occurs, the equivalent vector $\mathbf{r}(\mathbf{p}_{Di})$ of the original offset vector $\mathbf{V}(\mathbf{p}_i)$ is obtained for offsetting the corresponding parameter points. In the parameter space, where the parameter curve is a planar curve, the self-intersection in the offset polygon can be detected and eliminated efficiently using available algorithms (Hansen and Arbab, 1992; Park and Shin, 2002). As shown in Fig. 5.14, the local self-intersection in the offset polygon is detected and eliminated.



(a) (b)

Fig. 5.14 Self-intersection elimination in parameter space: (a) self-intersection; (b) eliminate self-intersection

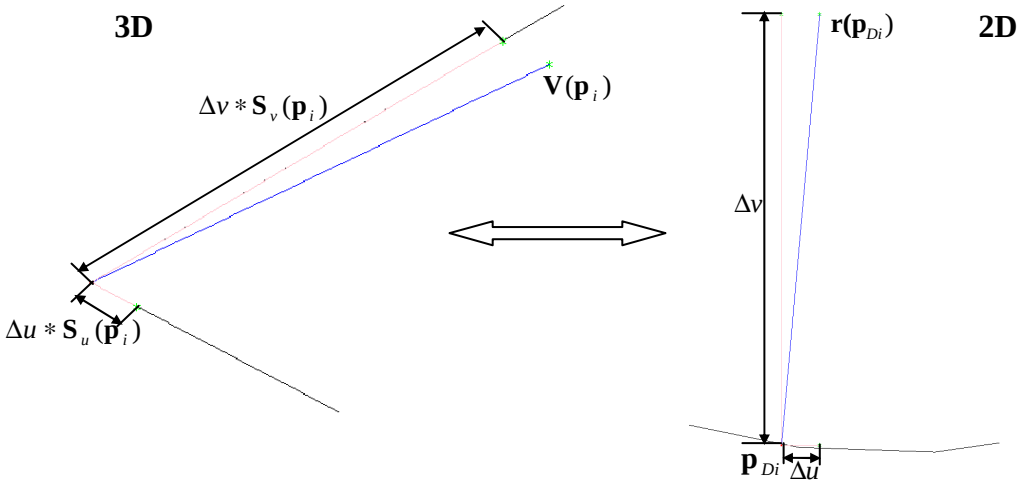


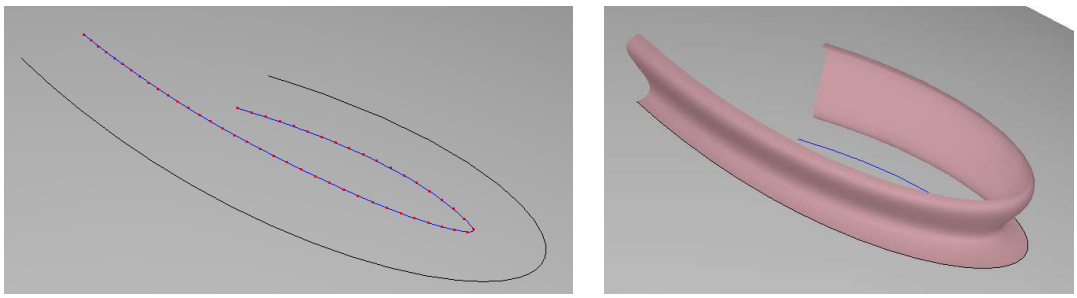
Fig. 5.15 Mapping between offset vectors on the tangent plane and parameter space

Once the self-intersection in the offset parameter curve has been eliminated, the remaining offset parameter points $\{\mathbf{p}_{Dj}\}$, $j = 1,...,m$, where m is the number of the remaining offset parameter points, are interpolated as a B-spline curve. Since the relative fair parameterization of the given curves is important for the smoothness of the blending surface (Cohen *et al*., 1997), the parameter values $\bar{u}_j$ of $\mathbf{p}_{Dj}$ are set using the centripetal method described next.

Let $d = \sum_{j=2}^{m} \sqrt{|\mathbf{p}_{Dj} - \mathbf{p}_{D(j-1)}|}$, then $\bar{u}_1 = 0$, $\bar{u}_m = 1$, $\bar{u}_j = \bar{u}_{j-1} + \frac{\sqrt{|\mathbf{p}_{Dj} - \mathbf{p}_{D(j-1)}|}}{d}$

$j = 2,...,m-1$.

In order to offset the original sample points $\{\mathbf{P}_i\}$, $i = 0,...,n-1$ on the tangent planes, it is necessary to obtain n equivalent offset vectors in the parameter space. Hence, the interpolating parameter curve is re-sampled with n points, and the offset parameter vector is formulated as $\mathbf{r}'(\mathbf{p}_{Di}) = \Delta u'\mathbf{u} + \Delta v'\mathbf{v}(0 \le i < n)$, where $\Delta u'$ and $\Delta v'$ are the new changes of u and v. Since the offset vectors in the parameter space and that in the tangent planes can be transformed from one another, as shown in Fig. 15, the offset vector $\mathbf{V}'(\mathbf{p}_i)$ in the corresponding tangent plane $\mathbf{T}_{S_k}$ can be reformulated as $\mathbf{V}'(\mathbf{p}_i) = \Delta u'\mathbf{S}_u(\mathbf{p}_i) + \Delta v'\mathbf{S}_v(\mathbf{p}_i)$. The original sample points $\mathbf{p}_i$ in the boundary curve are offset using the newly obtained direction vectors $\mathbf{V}'(\mathbf{p}_i)$ for surface blending, as shown in Fig. 5.16. Although $\mathbf{V}'(\mathbf{p}_i)$ is different from $\mathbf{V}(\mathbf{p}_i)$, the sample points in the boundary curve are still offset on the corresponding tangent planes, and hence the tangential smoothness is achieved. In this case, the new offset vector $\mathbf{V}'(\mathbf{p}_i)$ is not in the normal of the boundary curve any more, and the offset distances are not constant for all the sample points.



(a) (b)

Fig. 5.16 Blending surface after removal of self-intersection:
(a) offset boundary curve; (b) blending surface

5.3.3.2 Offset the Parameter Curve Directly.

If there is no self-intersection in the offset boundary curve, the offset vector $\mathbf{V}(\mathbf{p}_i)$ is normal to the boundary curve and the offset distance for all the sample points is

constant. Hence, the roundness along the boundary curve is constant. However, once the offset vector $\mathbf{V}(\mathbf{p}_i)$ is replaced by $\mathbf{V}'(\mathbf{p}_i)$, the sample points are not offset in the normal direction of the boundary curve, and the roundness along the boundary curve is not constant any longer. In practice, if it is necessary to keep the constant roundness along the boundary curve, the offset distance needs to be set carefully so that self-intersection does not occur. If the roundness radius along the boundary curve can be a variable, the offset vector $\mathbf{V}(\mathbf{p}_i)$ can be obtained by offsetting the parameter curve directly. In this case, the parameter curve is offset in the parameter space directly, and the self-intersection is detected and eliminated if needed. The offset vector in the parameter space can be mapped to the corresponding tangent planes, which has been illustrated in Fig. 5.15. As shown in Fig. 5.17, the parameter curve is offset to its normal direction by $d = 0.02$ with a tolerance of $\varepsilon = 10^{-5}$. The offset vectors are obtained in the parameter space, and mapped to the corresponding tangent planes. Thus, the offset curve in the tangent planes and the blending surface can be obtained, as shown in Fig. 5.18. Offsetting the parameter curve has the same effect with the method presented in Section 5.3.3.1, and it has lower computational cost. Hence, if the constant roundness is not a must and the boundary curve has small local curvature radius, which may probably cause self-intersection, offsetting the parameter curve directly is a better choice.

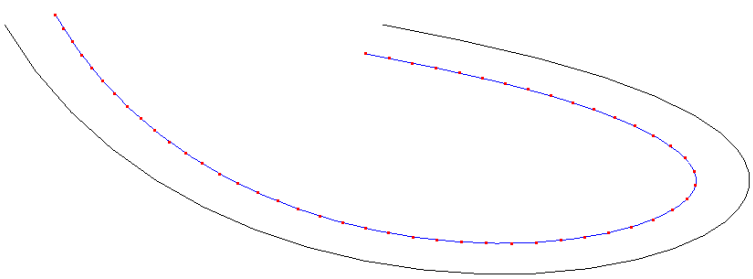
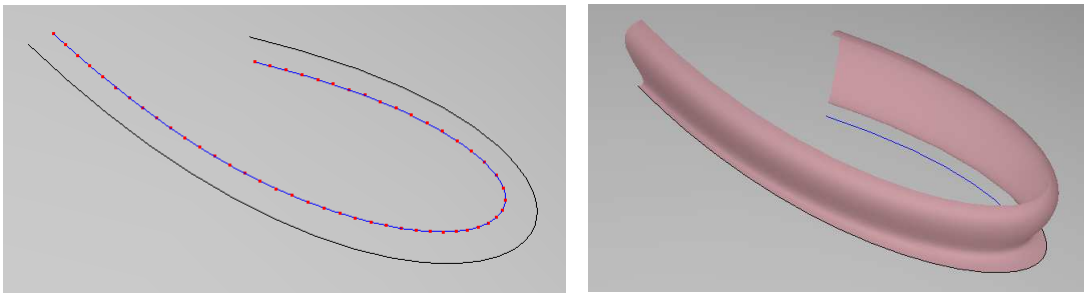


Fig. 5.17 Offset domain curve directly



(a) (b)

Fig. 5.18 Blending surface by offsetting the parameter curve directly: (a) offset boundary curve; (b) blending surface

5.3.4 Examples

In the proposed method, the blending surface is composed of certain Bézier patches, and hence the boundary curve of the displacement feature should be a Bézier curve or convertible to Bézier curves. Once the boundary curve is obtained in Bézier form, the tangent filed curve can be generated by interpolating the sampled points that are on the tangent planes of the base surface, and hence the blending surface patch can be constructed. The number of the sampled points depends on the complexity of the boundary curve, e.g. higher second derivative, and user-defined tolerance of the tangent field curve. As a result, the proposed method can be used for any boundary curve that is in Bézier form, and the computation time depends on the complexity of the boundary curve and user-defined tolerance of the tangent field curve. The surface blending algorithm presented in this section is validated using several examples. All of the examples have been implemented on an Intel(R) Pentium(R) D CPU 2.80GHz, 2G Memory, Microsoft Windows XP, Microsoft Visual C++ 6.0, and Open CASCADE.

In the first example, a B-spline curve in the parameter space is obtained by interpolating six points {(0.25, 0.375), (0.75, 0.375), (0.8125, 0.5), (0.75, 0.625), (0.25, 0.625), (0.1875, 0.5)}. The interpolating parameter curve is split into six Bézier curves,

and evaluated in a cubic ×quadratic Bézier surface using 3×4 control points, as shown in Fig. 5.19(a).

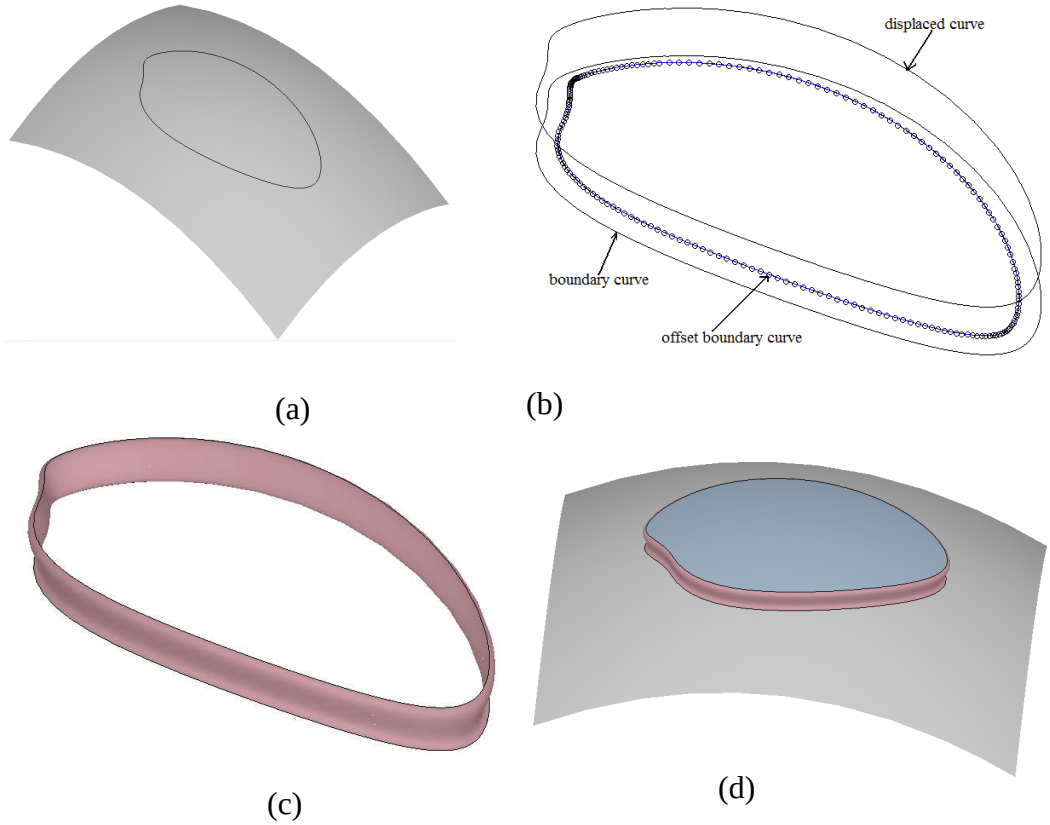


Fig. 5.19 Surface blending of a boundary curve#1: (a) boundary curve lying on the surface; (b) boundary curve, offset boundary curve and displaced curve; (c) blending surface; (d) displacement feature

The control points of the Bézier surface are given below.

$$\begin{cases} \mathbf{P}_{00} = (0,0,0)^T \\ \mathbf{P}_{10} = (3,0,2)^T \\ \mathbf{P}_{20} = (6,0,2)^T \\ \mathbf{P}_{30} = (9,0,0)^T \end{cases}, \begin{cases} \mathbf{P}_{01} = (0,3,2)^T \\ \mathbf{P}_{11} = (3,3,4)^T \\ \mathbf{P}_{21} = (6,3,4)^T \\ \mathbf{P}_{31} = (9,3,2)^T \end{cases}, \text{ and } \begin{cases} \mathbf{P}_{02} = (0,6,0)^T \\ \mathbf{P}_{12} = (3,6,2)^T \\ \mathbf{P}_{22} = (6,6,2)^T \\ \mathbf{P}_{32} = (9,6,0)^T \end{cases}.$$

The Bézier curves lying on the surface are offset by $d = 0.2$ with a tolerance of $\varepsilon = 10^{-5}$, and displaced towards the exterior of the surface by $h = 0.5$, as shown in Fig. 5.19(b). The blending surface is computed using the proposed algorithm, and shown in Fig. 5.19(c-d). Analogously, a depression displacement feature can be generated if the modified region is displaced towards the interior of the surface. To investigate the normal deviation across the boundary curve, 1500 points, which are equally spaced on

[0, 1], are sampled on each Bézier segment. The normal vectors at these sample points are determined in the given surface and in the blending surface respectively, and the normal vector deviation is calculated and averaged, as shown in Fig. 5.20.

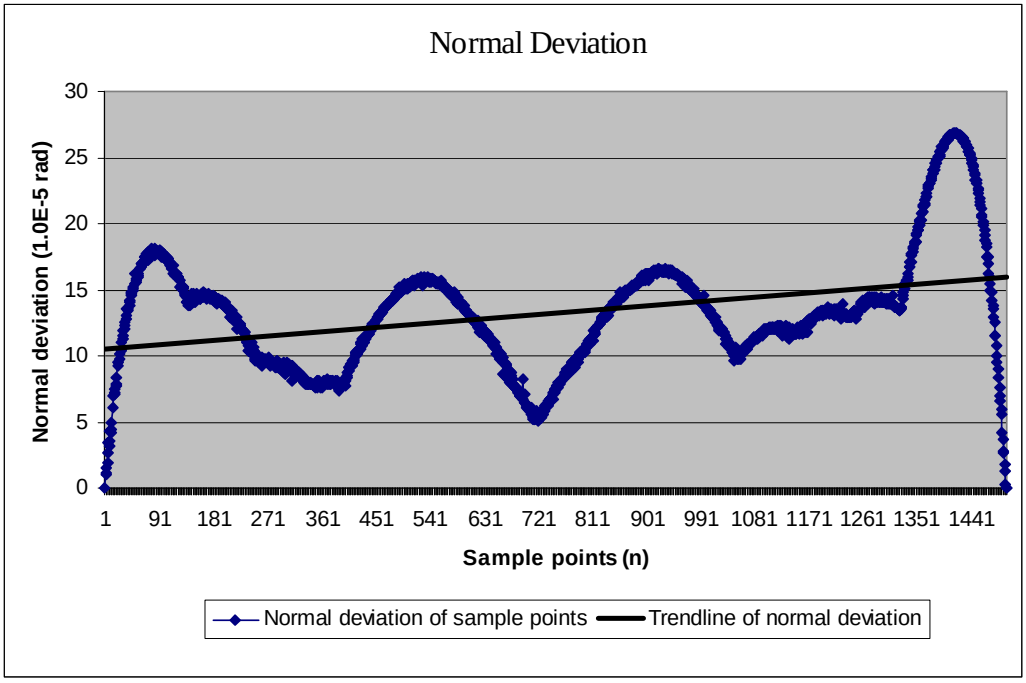


Fig. 5.20 Normal deviation across the boundary curve for the example in Figure 5.19

In the second example, a B-spline curve in the parameter space is obtained by interpolating eight points {(0.5, 0.7), (0.44, 0.67), (0.35, 0.55), (0.45, 0.33), (0.5, 0.36), (0.55, 0.33), (0.65, 0.55), (0.56, 0.67)}. The interpolating parameter curve is split into eight Bézier curves and evaluated in the given surface in example#1, as shown in Fig. 5.21(a). The Bézier curves lying on the surface are offset by $d = 0.3$ with a tolerance of $\varepsilon = 10^{-5}$, and displaced towards the exterior of the surface by $h = 0.5$, as shown in Fig. 5.21(b). When the displaced curve is offset, self-intersection problem arises, which are eliminated using the approach proposed in this research. The blending surface is computed using the proposed algorithm, as shown in Fig. 5.21(c-d). The normal vector deviation of 1000 sample points on each Bézier segment are calculated and averaged, as shown in Fig. 5.22.

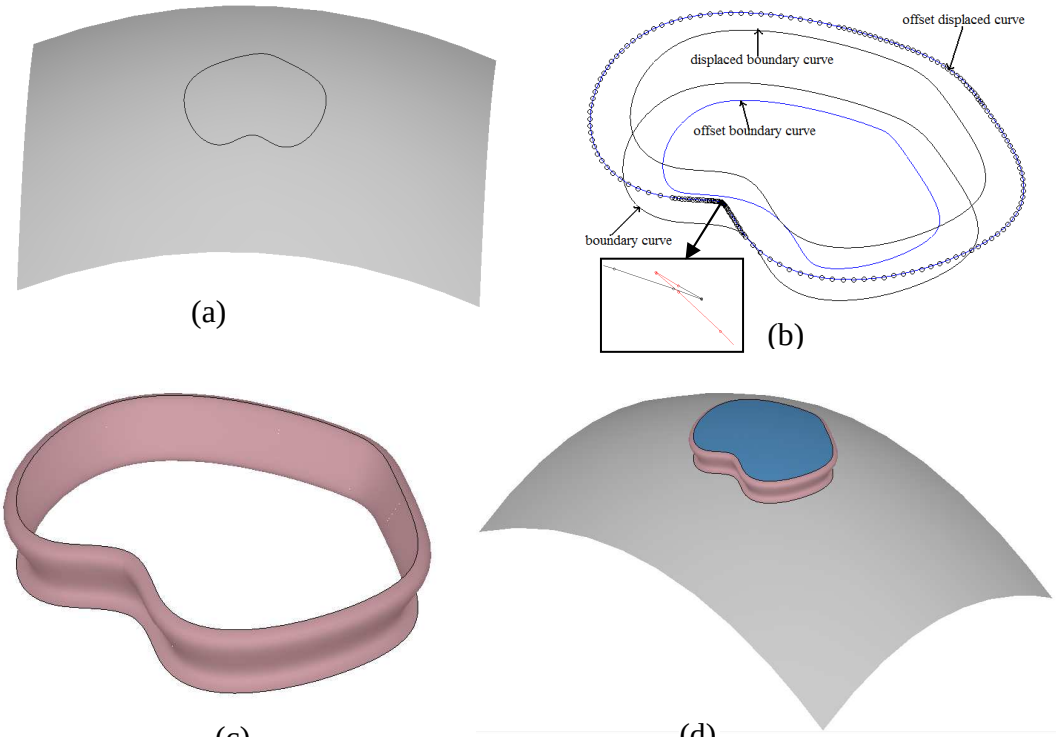


Fig. 5.21 Surface blending of a boundary curve#2: (a) boundary curve lying on the surface; (b) boundary curve, offset boundary curve, displaced curve and offset displaced curve; (c) blending surface; (d) displacement feature.

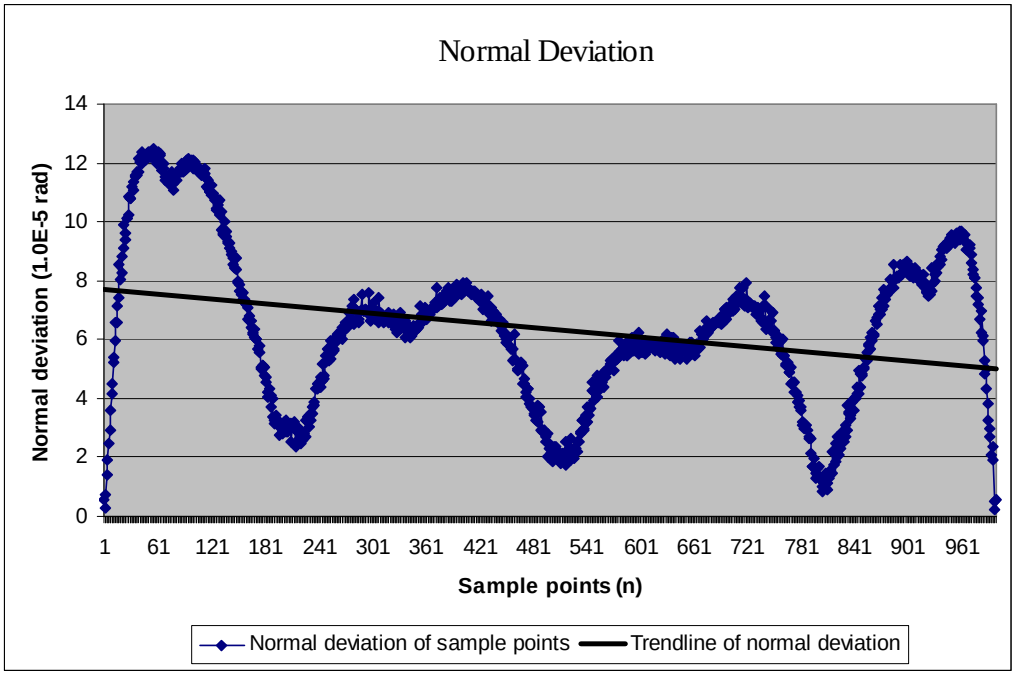


Fig. 5.22 Normal deviation across the boundary curve for the example in Figure 5.21

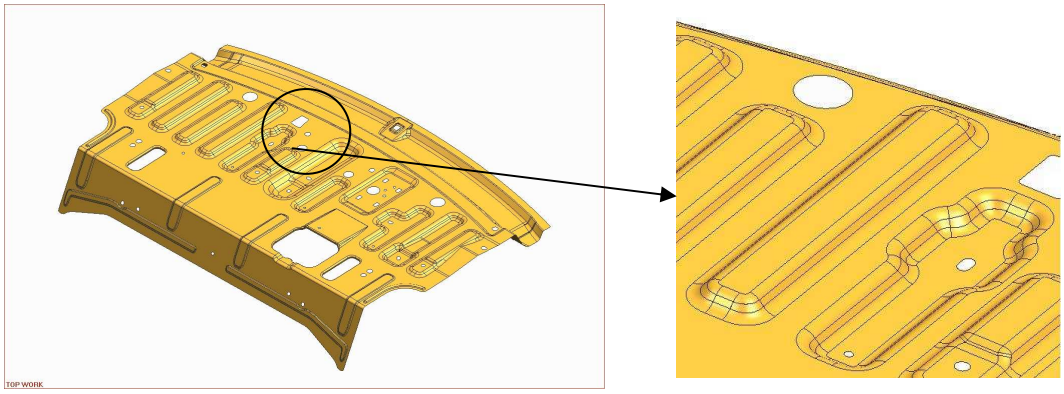


Fig. 5.23 Displacement features in a practical part

Displacement features, as shown in Fig 5.23, are commonly used in many practical products. In this study, one of the depression features is respectively generated using the proposed surface blending approach and Elsas's method, as shown in Fig. 5.24. It is impossible to observe the difference between the two methods from the visualization of the depression feature, so the normal deviation along the boundary curve, the number of the blending surface patches, and the computation time are summarized in Table 5.2. It should be noted that the user-specified tolerance of the tangent field curve is $\varepsilon = 10^{-4}$. In addition to displacement feature modeling, writing text on the parts can also be accomplished using this proposed approach. As shown in Fig. 5.25, the designers set the boundary of the required letter "LEI", and then the corresponding 3D letter can be generated by setting the parameters, in which the tangential smoothness across the boundary curves is maintained.

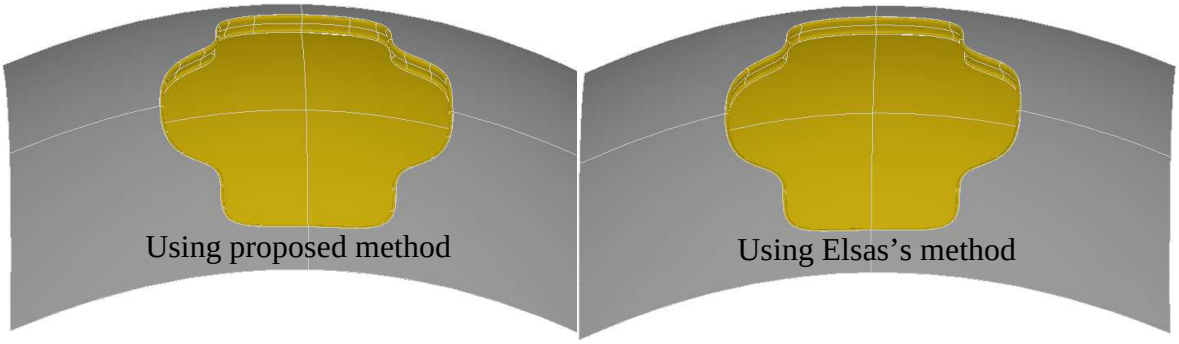


Fig. 5.24 Create displacement features using the proposed approach and Elsas's method

Table 5.2 Comparison between proposed method and Elsas's method

	Average normal deviation along boundary curve	Blending surface patches	Computation time
Elsas's method	2.2*1.0E-3	8	125ms
Proposed method	3.8*1.0E-5 (user-defined tolerance of tangent field curve is $\varepsilon = 10^{-4}$)	16	282ms

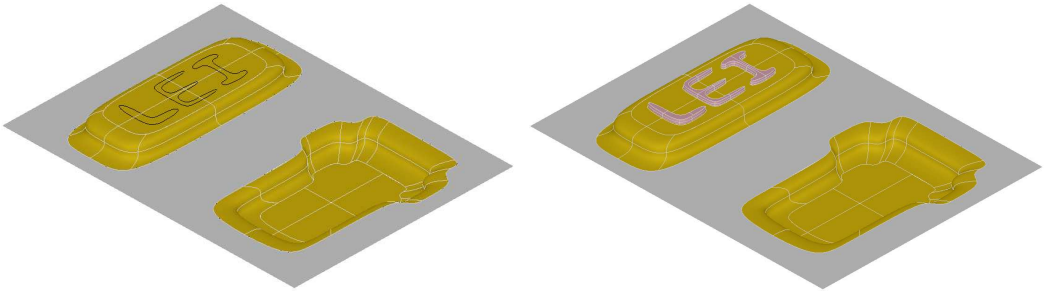


Fig. 5.25 Write texts on parts using the proposed approach

5.3.5 Summary

A surface blending approach for displacement feature modeling in freeform surfaces is presented in this work. To avoid the high polynomial degree of the tangent field curve obtained symbolically, an approximation for the Cubic Hermite Interpolant is proposed. It is found that the Bézier curve using the interior row of the control points in the blending surface should be in the tangent field. As a result, the boundary curve of the displacement feature is offset in the tangent field with a user-specified tolerance to obtain the interior control points of the blending surface. The local self-intersection problem in the offset curve can be transformed approximately and eliminated in the parameter space of the base surface. The proposed algorithm is validated with four examples, in which the boundary curve of the displacement feature can be specified flexibly by the users. The normal vectors along the boundary curve are determined in the blending surface and the base surface respectively, and this shows that the normal deviation is even smaller than the offset tolerance. Since the offset tolerance can be set

by the users for specific cases, the G^1 smoothness can be achieved with different user specified tolerance.

5.4 Displacement Feature Modeling in a Collaborative Environment

Collaborative feature modeling is a paradigm for the co-design of a product model, in which the scheduling of design activities and the product information sharing are the two crucial issues, as presented in the review Section 2.2.2. Since a team of designers co-create the design model, a granular-locking mechanism is proper for scheduling the concurrent design activities. In this section, the combination of displacement feature modeling with the collaborative feature modeling will be discussed, in which the co-ordination and model information sharing will be addressed.

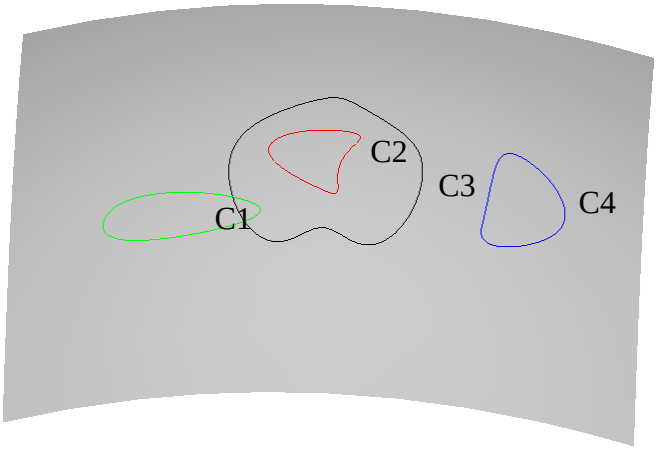


Fig. 5.26 Relationships of boundary curves

In displacement features, the feature shape is mainly determined by its boundary curve lying on the base surface. As a result, in a collaborative environment, the feature shape can be designed collaboratively by reviewing the boundary curve among the designers, after which the modified surface region is trimmed and the transition surface is generated. In addition, normally, several displacement features are embedded into a base surface for creating a functional part. The relationships between the boundary curves on a base surface can be categorized as interacting and non-interacting

relationships. The interacting relationships between a boundary curve that is defined on the base surface and a boundary curve already in the base surface include overlapping and nesting. As shown in Fig. 5.26, curves C_1 and C_3 are overlapping relationship, curves C_2 and C_3 are nesting relationship, and curves C_4 and C_3 are non-interacting relationship. Generally, the non-interacting boundary curves can be defined concurrently by designers, since their transition surfaces are normally non-interacting either. However, when the boundary of a feature nests in or overlaps with another boundary curve, the two boundaries can only be modified by one user at any time.

Based on the above classification, a granular locking mechanism can be used for scheduling the concurrent design activities in displacement feature modeling, as the mechanism presented in Chapter 4. The locking granularity can be a single feature that has no interacting boundary curve, or it can be a group of features whose boundary curves are interacting. As shown in Fig. 5.26, two groups of features are identified, namely, curves C_1 C_2 C_3 are in one group and C_4 is in another. A designer can only modify a displacement feature after receiving the permission from other designers. As such, designers can work on different features at the same time, and the design efficiency can be improved.

The change information of the design model needs to be synchronized, thus the replicated design models are maintained consistently. In this work, at the client sides, each user has the full-fledged modeling functions, thus only the modeling operation needs to be synchronized for updating product change. However, since a boundary curve defined by one user does not need to be defined and computed by other users again, the boundary curve is also transmitted each time for product information sharing.

A boundary curve is basically a B-spline curve or several Bézier curve segments, so it does not pose much communication load. When a boundary curve is reviewed and finalized by certain designers, it is broadcast to all sites together with the parameters for surface blending to update the designed model.

5.5 Summary

Freeform feature modeling has been discussed in this Chapter. Firstly, a simple volumetric freeform feature is created by standard sweeping operations, in which the profile 3D curve and the trajectory 3D curve are defined by interpolating certain 3D points. Secondly, the surface blending in displacement feature modeling is approximated to avoid the higher polynomial degree in the transition surface. In this approximation, the boundary curve is first offset in the tangent field, and then it is knot-refined to be compatible with the offset curve for surface blending. Thirdly, displacement feature modeling in a collaborative design environment is briefly discussed, where displacement features can be grouped according to the relationships of their boundary curves.

Chapter 6 Implementation Environment and Case Studies

The implementation methods and tools used in this study are described in this chapter. Firstly, the working environments for the case studies presented in previous chapters will be elaborated in details. Secondly, the proposed mechanisms are combined in the replicated collaborative feature modeling, where the modeling system is validated using two examples.

6.1 Implementation Studies

6.1.1 Open CASCADE Technology

Open CASCADE Technology (OCC) is a powerful open source C++ library, consisting of the classes and solutions in the areas of surface and solid modeling, 3D and 2D visualization, data exchange, etc. Modeling functions are used for constructing an object comprising of geometry and topology, and visualization functions are used to display and manipulate the designed object. As shown in Class#1, a solid box is being constructed by sweeping a sketch face along a vector, which is displayed in Fig. 6.1. Analogously, a freeform object, e.g., a Bézier surface, can be constructed and displayed, as described in Class#2. In OCC, all the modeling functions and data structure are carried out using C++, so it becomes more complicated when combining OCC with Java. In order to connect OCC and Java, Java Native Interface (JNI) can be used to provide the Interface functions. As coded in Class#3, the functions in Java can be defined as native functions, which are actually executed using the loaded DLL module. This DLL module is a library containing the modeling functions written in C++, as coded in Class#4. Once a native function in Java is called, the corresponding arguments and objects are transferred to the functions in the DLL module, which then returns back the calculated results to the Java Function.

```
Class#1
{
        //***********definition of the corner points in the sketch***************//
        gp_Pnt point1(0,0,0);
        gp_Pnt point2(1,0,0);
        gp_Pnt point3(1,1,0);
        gp_Pnt point4(0,1,0);
        //***********definition of the four edges in the sketch*****************//
        TopoDS_Edge edge1 = BRepBuilderAPI_MakeEdge(point1, point2);
        TopoDS_Edge edge2 = BRepBuilderAPI_MakeEdge(point2, point3);
        TopoDS_Edge edge3 = BRepBuilderAPI_MakeEdge(point3, point4);
        TopoDS_Edge edge4 = BRepBuilderAPI_MakeEdge(point4, point1);
        //***************the sketch of the solid box*************************//
        TopoDS_Wire sketch = BRepBuilderAPI_MakeWire(edge1, edge2, edge3,
                                                     edge4);
        TopoDS_Face sketchFace = BRepBuilderAPI_MakeFace(sketch);
        //***************  sweeping  solid  box*******************************//

        gp_Vec vec(0, 0, 1.5);
        TopoDS_Shape box = BRepPrimAPI_MakePrism(sketchFace, vec);
        //***************visualization of the solid box***********************//
        Handle(AIS_Shape) ais_shape = new AIS_Shape(box);
        myAISContext->SetColor(ais_shape,Quantity_NOC_PINK);
        myAISContext->SetMaterial(ais_shape,Graphic3d_NOM_PLASTIC);
        myAISContext->SetDisplayMode(ais_shape,1);
        myAISContext->Display(ais_shape);
}
```

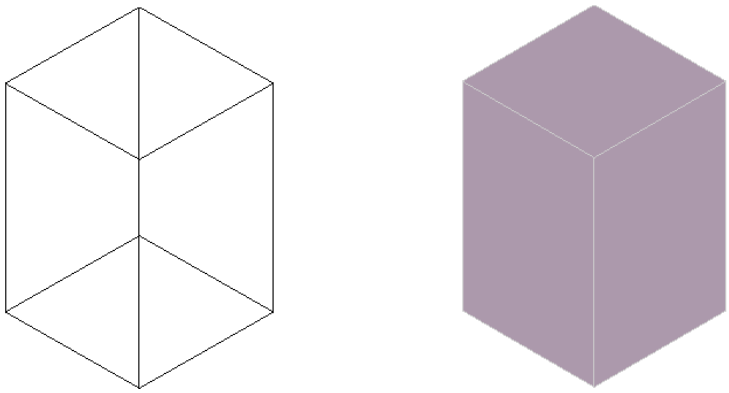


Fig. 6.1 Visualization of a solid box shape

```
Class#2
{
        //***************control points of the Bézier surface*****************//
        TColgp_Array2OfPnt Poles(0, 3, 0, 2);
        Poles.SetValue(0, 0, gp_Pnt(0,0,0));
        Poles.SetValue(1, 0, gp_Pnt(3,0,2));
        Poles.SetValue(2, 0, gp_Pnt(6,0,2));
        Poles.SetValue(3, 0, gp_Pnt(9,0,0));
```

```
        Poles.SetValue(0, 1, gp_Pnt(0,3,2));
        Poles.SetValue(1, 1, gp_Pnt(3,3,4));
        Poles.SetValue(2, 1, gp_Pnt(6,3,4));
        Poles.SetValue(3, 1, gp_Pnt(9,3,2));
        Poles.SetValue(0, 2, gp_Pnt(0,6,0));
        Poles.SetValue(1, 2, gp_Pnt(3,6,2));
        Poles.SetValue(2, 2, gp_Pnt(6,6,2));
        Poles.SetValue(3, 2, gp_Pnt(9,6,0));
        //********************geometry of the Bézier surface*****************//
        Handle(Geom_BezierSurface) mySurf = new Geom_BezierSurface(Poles);
        //********************topology of the Bézier surface*****************//
        TopoDS_Face Face = BRepBuilderAPI_MakeFace(mySurf);
        //********************visualization of the Bézier surface**************//
        Handle(AIS_Shape) ais_shape = new AIS_Shape(Face);
        myAISContext->SetColor(ais_shape,Quantity_NOC_GRAY);
        myAISContext->SetMaterial(ais_shape,Graphic3d_NOM_PLASTIC);
        myAISContext->SetDisplayMode(ais_shape,1);
        myAISContext->Display(ais_shape);
}

Class#3
public class MakeSegment
{
        static
        {
                System.loadLibrary("geometryJni"); //the DLL module//
        }
        public GC_MakeSegment(gp_Pnt P1, gp_Pnt P2)
        {
                GC_MakeSegment_0(P1, P2);
        }
        //*****************native function*****************************//
        public final native void GC_MakeSegment_0(gp_Pnt P1, gp_Pnt P2);
}

Class#4
JNIEXPORT void JNICALL Java_geometryJni_GC_1MakeSegment_GC_1M
        akeSegment_10 (JNIEnv *env, jobject theobj, jobject P1, jobject P2)
{
        //********P1 and P2 are the 3D points transferred from Java function******//
        gp_Pnt* the_point1 = (gp_Pnt*) jcas_GetHandle(env,P1);
        gp_Pnt* the_point2 = (gp_Pnt*) jcas_GetHandle(env,P2);
        GC_MakeSegment*  theret  =  new  GC_MakeSegment(*the_point1,
*the_point2);
        //********returns back the Segment constructed using OCC functions******//
        jcas_SetHandle(env, theobj, theret);
}
```

6.1.2 Implementation Methods for History-Independent Modeling

The performance measurement of the proposed history-independent modeling, which has been presented in Chapter 3, was conducted using OCC and VC++. In this section, only the average behavior model, introduced in Fig. 3.11(b), will be studied to measure the performance of the proposed modeling approach, and the performance measurements of the other two representative models are presented in Appendix A. This average behavior model consists of a *Block* with a row of 33 feature groups, each of which have three intersecting features inserted sequentially: first *Rib*, then *Slot*, and finally *ThroughHole*, as described in Class#5 and displayed in Fig. 6.2. The 'remove operation' of a *Rib* is described in Class#6 and is displayed in Fig. 6.3, in which there are three steps, namely, removing the boundary faces originating the *Rib*, merging the intersection face portions stored at this step, and updating the boundary faces of the intersecting features created later than *Rib*. The intersection face portion of the *Rib*, as shown in Fig. 6.4, is identified and stored during its 'add operation'.

```
Class#5
{
        TopoDS_Shape result, rib, slot, hole;
        TopoDS_Shape block= Block(0,0,0,665,20,20);
        result = block
        clock_t start, finish;
        for(int i=0;i<33;i++)
        {
                rib = Rib(10+i*20,0,20,20,10,5,30);
                slot = Block(5+i*20,0,0,10,20,5);
                hole = Hole1(15+i*20,10,0,3,25);
                BRepAlgoAPI_Fuse fuse(result, rib);
                BRepAlgoAPI_Cut cut(fuse.Shape(),slot);
                start=clock();
                BRepAlgoAPI_Cut cut1(cut.Shape(),hole);
                result = cut1.Shape();
                finish= clock();
                //*****************measure the modeling time****************//
                double time =double(finish-start)/CLOCKS_PER_SEC;
                myAverageModel = result;
        }
}
```

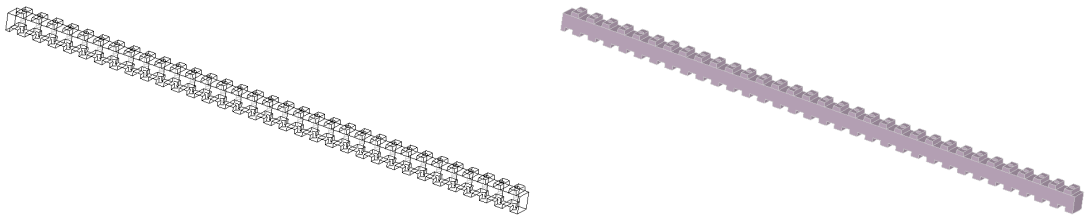


Fig. 6.2 Average behavior model

```
Class#6
{
        BRepBuilderAPI_Sewing sewing;
        int j=20; //suppose the 20th Rib is being removed//
        TopoDS_Shape result = averageModel; //load the designed average model//

        rib = Rib(10+j*20,0,20,20,10,5,30);
        hole = Hole1(15+j*20,10,0,3,25);
        //**********find the face originating from the rib being removed******//
        int n=0;
        BRepAlgoAPI_Common common(result,rib);
        Handle(TopTools_HArray1OfShape) myArrayRemoveFace;
        TopTools_ListOfShape listOfShape;
        for(TopExp_Explorer exp(result, TopAbs_FACE);exp.More();exp.Next())
        {
                listOfShape = common.Modified(exp.Current());
                if(listOfShape.Extent())
                        myArrayRemoveFace->SetValue(n++,exp.Current());
        }
        //*******end of find the face originating from the rib being removed****//

        TopoDS_Shape interFace;
        BRep_Builder bb;
        CString tempCS = "averagecase/H"+"j+1"+"Int.brep";
        BRepTools::Read(interFace,path,bb); //load the stored 'intersection face
portion' //
        BRepTools_ReShape reShape;
        start = clock();
        //**********remove the face originating from rib******************//
        for(int i=0;i<5;i++)
                reShape.Remove(myArrayRemoveFace->Value(i));
        result = reShape.Apply(result);
        //***********end of remove the face originating from rib*************//
        sewing.Add(result);
        //************update the intersection face portion******************//
        sewing.Add(interFace);
        sewing.Perform();
        result = sewing.SewedShape();
        //***************end of update the intersecting face*****************//
```

```
        //***************update the face of the intersecting feature***********//
        BRepAlgoAPI_Common common1(rib,hole);
        result = BRepAlgoAPI_Cut(result,common1.Shape());
        //***************end of update the face of the intersecting feature******//
        finish = clock();
        //***************measure the modeling time**********************//
        double time =double(finish-start)/CLOCKS_PER_SEC;
}
```

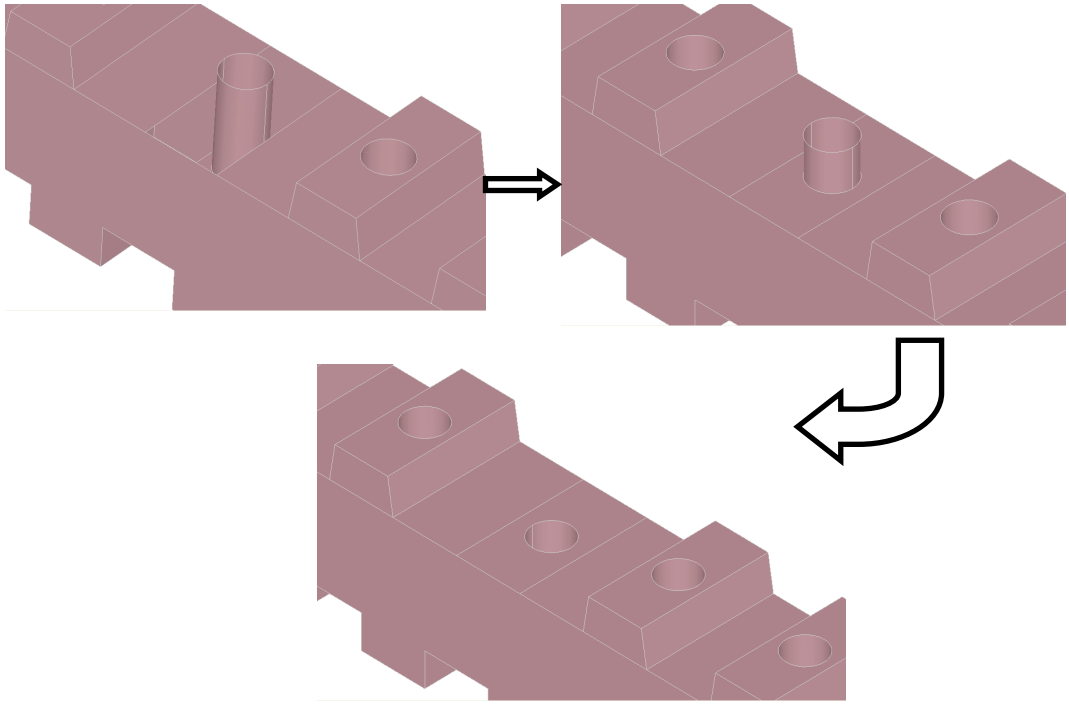


Fig. 6.3 Proposed 'remove feature' operation

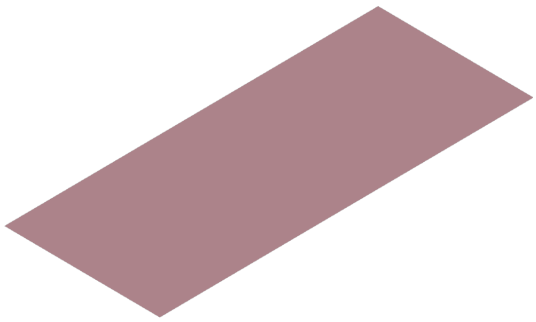


Fig. 6.4 Intersection face portion of the *Rib*

6.1.3 Maple used in Displacement Feature Modeling

Maple is a computer algebra system, and it has extensive support for numeric computations to a precision which can be set arbitrarily, as well as symbolic computation and visualization. In this study, Maple is used for the numeric and

symbolic computations in the modeling of freeform curves and surfaces, such as evaluating a 2D parametric curve in a 3D surface, calculating the derivatives of curves and surfaces, etc. Since B-spline basis functions cannot be represented directly in Maple, the B-spline curve used in this work is converted into Bézier segments to enable the symbolic calculation. For the evaluation of a 2D parametric curve, the obtained result is a 3D space curve in polynomial representation, which can be converted to a Bézier curve using the available algorithm. As described in Algorithm#1, K is the matrix for the change of basis from the power basis to the Bernstein basis, A are the coefficients of the original polynomial, and C are the control points of the Bézier curve. Once the sample points and offset points are obtained in Maple, they are interpolated as B-spline curves in VC++ for surface blending.

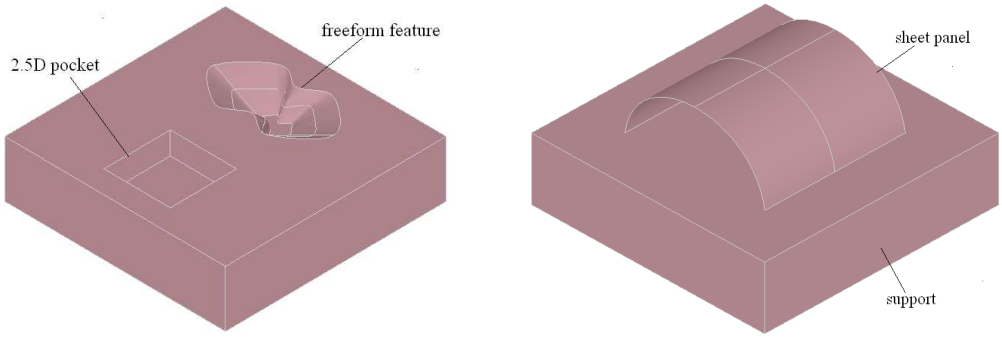
Algorithm#1

```
restart : alias(C = binomial) :
d := (n, i, j) → C(i, j) / C(n, j);
 n := 15;
K := Matrix(1 ..n + 1, 1 ..n + 1) :
for i to n + 1 do
  for j to n + 1 do
    K[i, j] := eval( d(n, i−1, j−1));
  end do;
end do;
K;
C := K.A ;
```

6.2 Case Studies

In current product design and aesthetic design, both regular-shaped features and freeform features are commonly used. In this section, two types of product models are used to validate the proposed modeling system. In the first case model, a freeform feature has the similar representation as a regular 2.5D feature, comprising a top surface, a bottom surface, and the transition surface. However, the transition surface

and bottom surface of the freeform feature are described in freeform representation, including a single or multiple patches (Sundararajan and Wright, 2004), but the bottom surface of the regular 2.5D feature is a planar surface. As shown in Fig. 6.5(a), the 2.5D pocket has a planar bottom surface, but the transition surface of the freeform feature comprises multiple freeform surface patches. In the second case study model, the product comprises two parts, namely the support part and the sheet panel, as shown in Fig. 6.5(b). The support part is normally composed of regular features, whilst the sheet panel is a freeform surface including depression and protrusion surface regions.



(a) (b)

Fig. 6.5 (a) Freeform feature and 2.5D feature, (b) support part and sheet panel part

A proof-of-concept prototype system of the proposed modeling approach has been established. The server manages a design session, containing some Java Socket, HTTP and Java RMI services, as shown in Fig. 6.6. Through the socket services, designers can download the needed modeling kernel from server. The HTTP and RMI services enable designers to obtain the exported remote functions on server and communicate with the server through design events. The design events in this work include design operations, e.g., ‘create feature’ operation and ‘modify feature’ operation, and communication message. On the client sides, the design context contains the feature model, including the features and feature relationships, and the resulting geometric

model. The viewing, manipulation and modeling functions are implemented based on OCC, and as shown in Fig. 6.7.

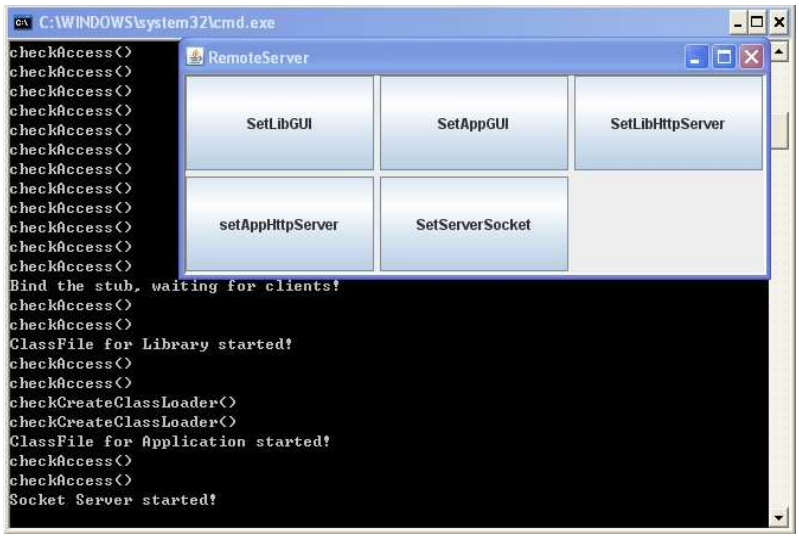


Fig. 6.6 Remote server

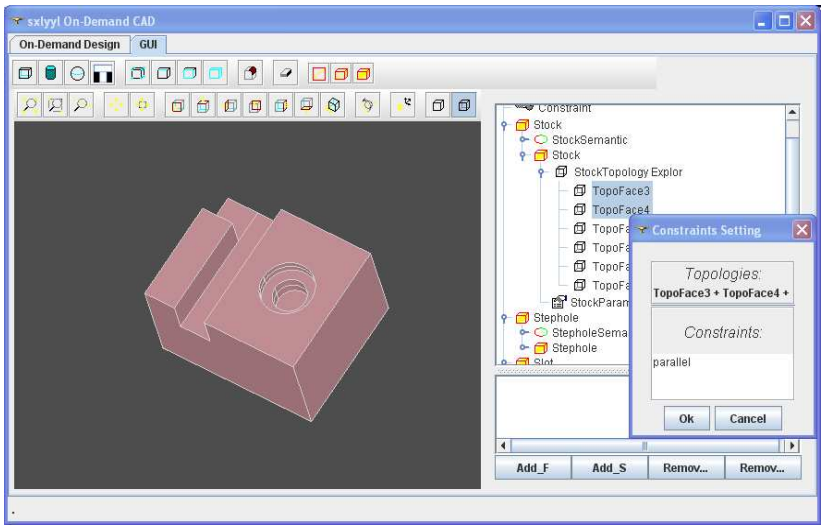


Fig. 6.7 Design context

6.2.1 First Case Study

Two designers A and B are working on a product model concurrently, as shown in Fig. 6.8. Designer A intends to add two pins on the bottom surface of the *pocket* , and sends a ‘create feature’ operation event to the server, described as

$RT(CF) < S_A, (pocket), (feature\#10,10,10,30, face1), 9 >$.

The specification of the feature operation includes its identity *feature*#10 , the parameters (10,10,30) , and the reference entity *face*1. Meanwhile, designer B intends to add a freeform pocket in the top surface of the *stock* , described as

$RT(CF) < S_B, (stock), (\ feature\#10,\ freeformDesign,\ face2), 9 >$.

Once the server receives and broadcasts the two operation events, designer B finds that the operation performed by designer A should be executed first due to his lower priority. As a result, in the resulting model, the pins defined by A have the identity *feature*#10 , and the freeform feature defined by B has the identity *feature*#11.

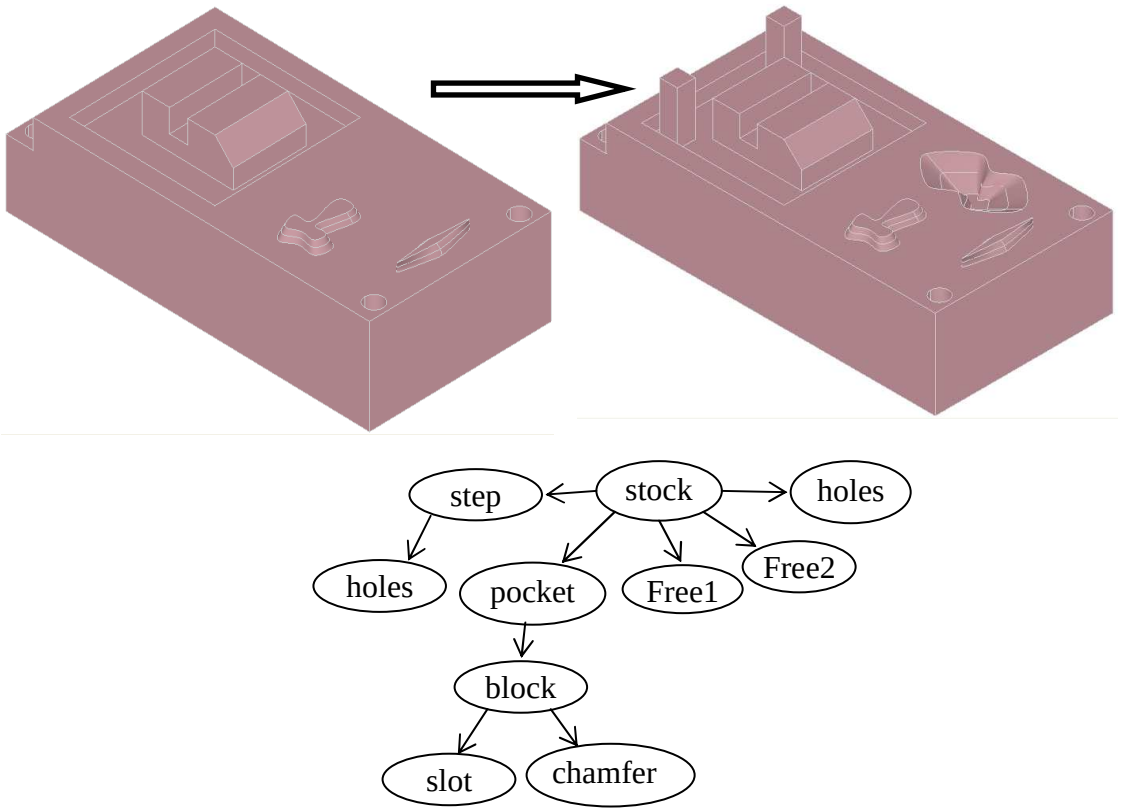


Fig. 6.8 Case model#1

6.2.2 Second Case Study

In the second case study, a team of designers work on a product model including a support part and a sheet panel, as shown in Fig. 6.9. In this case, the design team can be divided into two working groups. One group works on the support part, and the other focuses on the sheet panel. This modeling paradigm is similar to assembly design,

and the interface between the support part and the sheet panel is predefined and constrained. As such, the concurrent operations on the support part can be managed using the locking mechanism presented in Chapter 4, and the operations on the sheet panel are managed using the mechanism presented in Section 5.4.

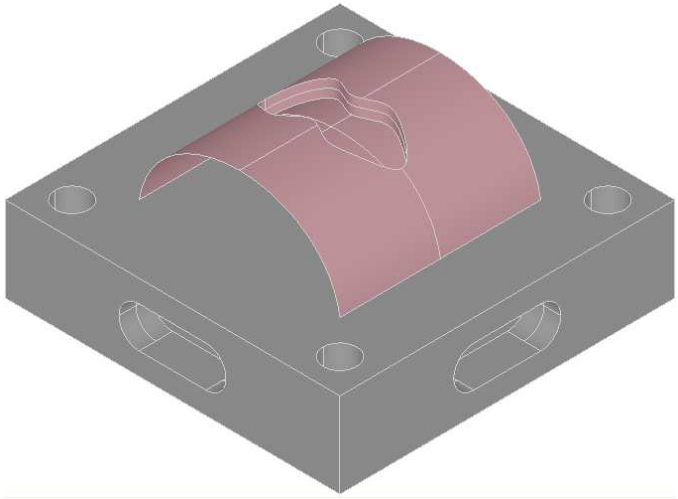


Fig. 6.9 Case model#2

6.3 Summary

The implementation tools and the programming works used in this study are presented in this chapter. A proof-of-concept system used for validating the proposed modeling approach has been established based on Java, VC++ and OCC. Two types of models that can be used in the proposed collaborative platform are introduced, namely the model including regular 2.5D features and freeform features, and the model comprising of a support part and a sheet panel.

Chapter 7 Conclusions and Future Work

7.1 Conclusions and Contributions

The primary objective of this research is to develop a set of methodologies to provide a productive and effective collaborative environment for product modeling, in which a team of designers work together on creating a regular prismatic model or designing displacement features on freeform surfaces. The investigated and explored works include: a history-independent modeling approach for overcoming the flaws of the boundary evaluation in history-based modeling, a granular locking mechanism for providing a parallel working process, and a collaborative environment for creating displacement features on freeform surfaces.

7.1.1 Collaborative Feature Modeling Framework

The proposed design framework is a replicated collaborative feature modeling system. On the client sides, two modeling functions were enhanced, namely, a history-independent modeling approach for prismatic models and a surface blending approach for displacement feature modeling. On the server side, a granular locking mechanism was explored for scheduling the concurrent design operations, and the product information can be synchronized by broadcasting the modeling operations across the designers.

This modeling platform provides a valuable paradigm for designers working together on a complex product model, which is strongly needed in current product development. In this case, a group of designers work on different portions of a part model, so as to achieve the design task concurrently. Meanwhile, the engineers in different domains

can cooperate on the definition of a feature shape before its execution, which ensures that the design model satisfies more constraints than stand-alone modeling. As presented in Chapter 6, a complex product model, including both regular features and freeform features, can be accomplished collaboratively in this proposed system.

7.1.2 Proposition of a History-Independent Modeling Approach

This study explored the history-independent modeling for overcoming the shortcomings of the traditional history-based modeling, as presented in Chapter 3. The 'remove feature' operation is accomplished in three steps: firstly, the boundary faces originating from the feature being edited are removed; secondly, the boundary contribution of its intersection face portion is updated; lastly, the boundary contribution of its intersecting features is updated. Since the creation step of the feature being modified is changed after each modification, the problems caused by the static 'feature creation order' can be solved. It is found that the computational complexity of the boundary evaluation using the proposed approach is better than that in history-based modeling. This is because all the features are re-evaluated sequentially in history-based modeling, but only the intersecting features of the feature being edited are re-evaluated in this work. The simulation results for three representative models show that more computation time is needed compared to the work reported by Bidarra *et al*. (2005), which is due to that not all partitioned faces are stored in this work. This approach takes a major step towards 'history-independent modeling', in which the feature model is always evaluated according to designer's specifications and the computation efficiency is improved. All the topological entities of the current B-rep model can be referred to constrain the feature being modified, and the re-evaluation is on the basis of the current status of the boundary faces. As such, the feature being

modified can be specified and evaluated accurately according to the desirable intentions. The modeling mechanism used in current feature-based modeling may be replaced by the proposed mechanism so as to solve the problems encountered in practical works.

7.1.3 Enhancement of the Granular Locking Mechanism for Replicated Collaborative Feature Modeling

In this study, the granular locking mechanism was enhanced so as to address two issues, namely, maintaining the exclusive 'feature creation order', solving the potential operation conflicts, as presented in Chapter 4. It is found that the potential conflicts of design operations caused by feature interactions can be resolved by the correspondence of the modified topological entities, in which the modified topological faces are tracked using a *FaceIdGraph* and the modified topological edges are identified using their adjacent faces. As such, all the operations may be executed correctly and the consistency of the replicated models would be maintained. Compared to the works reported by Li *et al*. (2008) and Jing *et al*. (2009), this work has the advantage in that the operation conflicts are resolved by the system automatically and the replicated models are synchronized consistently. Hence, this work extends the previous works on using the granular locking mechanism in collaborative feature modeling. In this work, the designers can perform operations at the same time, and the concurrent operations are coordinated and executed by the modeling server.

7.1.4 Proposition of a Surface Blending Approach for Creating Displacement Features in Freeform Surfaces

The freeform feature modeling implemented in a collaborative design environment has been explored in this work. Specifically, the modeling procedure of displacement

features was discussed, including the specification of the boundary curve and the surface blending. As presented in Chapter 5, a surface blending approach for approximating the Cubic Hermite Interpolant was proposed and validated. It is found that the tangential smoothness across the boundary curve can be achieved by offsetting the boundary curve in its tangent field, and then constructing the transition surface using the control points of the curves obtained. As such, the blending surface has a lower polynomial degree than that obtained using standard Cubic Hermite Interpolant, in which the tangent field curves are computed symbolically. It is because the polynomial degree of the offset curve depends solely on the interpolating algorithm, which can provide much lower-degree B-spline curves. In symbolic computation, however, the tangent curve of a boundary curve of n degree can be as high as $(2nm-1)^3$, where m is the degree of the base surface in u and v directions. In addition, the proposed blending approach can achieve tangential smoothness for a more complex boundary curve, which is quite useful in practice, compared to the works reported by van Elsas and Vergeest (1998). The investigation of the normal deviation along the boundary curve indicates that the normal deviation is even smaller than the offset tolerance. This shows the proposed approximation approach has good accuracy, and it provides a valuable approach for surface blending in practice.

In this proposed approach, users can offset the boundary curve with different tolerances for specific applications, which should provide flexibility for displacement feature modeling. In conceptual design, the accuracy of the smoothness may be not critical, so designers can offset the boundary curve with a large tolerance, which would not affect the visual effect of the designed model. In detailed design, a smaller tolerance can be used for offsetting the boundary curve, which generates a blending

surface that has better smoothness across the boundary curve and has lower polynomial degree. In addition, the approximation approach may be extended for achieving higher smoothness across the boundary curve.

7.2 Future Works and Suggestions

7.2.1 Development of History-Independent Modeling

Being an exploratory and preliminary study, the proposed history-independent modeling approach needs more research efforts in several issues, such as the naming and matching of topological entities, database management, position referencing of features, and the design of a graphical user interface, etc. The boundary faces are stored and retrieved frequently in this work, but the management of the boundary faces has not been explored. The naming and matching mechanism used in this work is adapted from the reported works (Capoyleas *et al*., 1996; Cripac, 1997; Wu *et al*., 2001; Wang and Nnaji, 2005), and it is not implemented. Furthermore, the matching of boundary entities are lacking, as the correspondence of the reference faces and edges presented in Chapter 4. An intelligent mechanism for the correspondence of boundary entities is very useful in the modeling fields, such as solving the persistent naming problem, compatible exchange of two models represented in different design systems, etc. Consequently, the naming and matching mechanism would need more research efforts in future, both in algorithm and in implementation.

7.2.2 Exploration in Freeform Feature Modeling

7.2.2.1 Evaluation of a 3D Curve lying on a Freeform Surface

The boundary curve of the displacement features was calculated symbolically using Maple in this work, which has a higher degree as presented in Section 5.3.1. The

approximation algorithm proposed by Yang *et al*. (2008) decreases the polynomial degree, but it generates many curve segments, which makes surface blending very complicated. As a result, a good alternative algorithm is needed for generating a boundary curve with a lower degree. As it is known, the degree of a 3D curve lying on a base surface depends on both the degree of its parameter curve and the degree of the base surface. In order to reduce the degree of the parameter curve, Yang *et al*. (2008) used a polyline to approximate the parameter curve, which results in many line segments in the evaluated 3D curve. Since the degree of the parameter curve cannot be reduced lower than a polyline, the algorithms from this angle would not produce a desirable solution. Hence, the boundary curve should be approximated during its evaluation on the freeform surface, as presented by Renner and Weiβ (2004). The key point is to provide an algorithm that can be used with different user-specified tolerances.

7.2.2.2 Surface Blending in Displacement Feature Modeling

In this work, an approximation approach was proposed for surface blending, and it can achieve the tangential smoothness across the boundary curve. The boundary curve needs to be offset in the tangent field, which is an offset issue of a 3D space curve. Offset curve in 3D space is useful in practice, but, currently, the offset curve is mainly addressed in the 2D domain, as the studies reported in the literature (Pekerman *et al*., 2008; Seong *et al*., 2006). As a result, an effective algorithm for offsetting a 3D curve is needed in future work. Specifically, the self-intersection issue in the offset curve should be addressed as well, since both local and global self-intersections are common in offset curves.

The surface blending approach used in this work only achieves G^1 smoothness across the boundary curve, which is not sufficient in practice. The G^n smoothness may be required in current surface modeling. Consequently, a more effective algorithm is needed in future work for achieving the G^n smoothness. The expected algorithm may be an approximation as the approach presented in this work, but the key point is that the algorithm should be implemented with different user-specified tolerances. Once a more useful surface blending has been implemented, the surface modeling of a freeform model could become intuitive and effective.

References:

Abrahamson, S., D. Wallace, N. Senin and P. Sferro. Integrated design in a service marketplace. Computer-Aided Design, 32, pp.97-107. 2000.

Bidarra, R., K. Jan De Kraker and W. F. Bronsvoort. Representation and management of feature information in a cellular model. Computer-Aided Design, 30(4), pp.301-313. 1998.

Bidarra, R. and W.F. Bronsvoort. Semantic feature modelling. Computer-Aided Design, 32, pp.201-225. 2000.

Bidarra, R., E. van den Berg and W.F. Bronsvoort. A Collaborative Feature Modeling System. Journal of Computing and Information Science in Engineering, 2, pp.192-198. 2002.

Bronsvoort, W. F. and A. Noort. Multiple-view feature modelling for integral product development. Computer-Aided Design, 36, pp.929-946. 2004.

Bidarra, R., J. Madeira, W.J. Neels and W.F. Bronsvoort. Efficiency of boundary evaluation for a cellular model. Computer-Aided Design, 37, pp.1266-1284. 2005

Cavendish, J.C. and S.P. Marin. A procedural feature-based approach for designing functional surfaces. In Proc. Topics in Surface Modeling, H. Hagen (ed), Geometric Design Publications, Philadelphia, 1992, pp.145-168.

Cavendish, J.C. Applications-Integrating feature-based surface design with freeform deformation, Computer-Aided Design, 27, pp.703-711. 1995.

Chen, X.P. and C.M. Hoffmann. On editability of feature-based design. Computer-Aided Design, 27(12), pp.905-914. 1995.

Capoyleas, V., X.P. Chen and C.M. Hoffmann. Generic naming in generative, constraint-based design. Computer-Aided Design, 28(1), pp.17-26. 1996.

Cripac, J. A mechanism for persistently naming topological entities in history-based parametric solid models. Computer-Aided Design, 29(2), pp.113-122. 1997.

Cohen, S., G. Elber and R. Bar-Yehuda. Matching of freeform curves. Computer-Aided Design, 29, pp.369-378. 1997.

Chan, S.C.F. and V.T.Y. Ng. Real-Time Collaborative Solid Shape Design (RCSSD) on the Internet. Concurrent Engineering, 10(3), pp.229-238. 2002.

Chen, L., Z.J. Song and L. Feng. Internet-enabled real-time collaborative assembly modeling via an e-Assembly system: status and promise. Computer-Aided Design, 36, pp.835-847. 2004.

de Carmo, M. Differential geometry of curves and surfaces. Prentice-Hall. 1976.

Duan, W., J. Zhou and K. Lai. FSMT: a feature solid-modeling tool for feature-based design and manufacture. Computer-Aided Design, 25(1), pp.29-38. 1993.

Du, D.Z. and F. Hwang (ed). Computing in Euclidean Geometry. pp.266-298, Singapore: World Scientific. 1995.

Durand, C. and C.M. Hoffmann. A systematic framework for solving geometric constraints analytically. Journal of Symbolic Computation, 30, pp.493-519. 2000.

Ding, L., D. Davies and C.A. McMahon. The integration of lightweight representation and annotation for collaborative design representation. Research in Engineering Design, 19, pp.223-238. 2009.

Elber, G. Generalized filleting and blending operations toward functional and decorative applications. Graphical Models, 67, pp.189-203. 2005.

El-Tayeh, A., N. Gil and J. Freeman. A methodology to evaluate the usability of digital socialization in “virtual” engineering design. Research in Engineering Design, 19, pp.29-45. 2008.

Farouki, R.A.M.T., R. Sverrisson. Approximation of rolling-ball blends for free-form parametric surfaces. Computer-Aided Design, 28 pp.871-878. 1996.

Fontana, M., F. Giannini and M. Meirana. A free form feature taxonomy. In Proc. Eurographics’99, P. Brunet and R. Scopigno (Eds), 18(3), 1999.

Flöry, S. and M. Hofer. Constrained curve fitting on manifolds. Computer-Aided Design, 40, pp.25-34. 2008.

Gerhard, J.F., D. Rosen, J.K. Allen and F. Mistree. A Distributed Product Realization Environment for Design and Manufacturing. Journal of Computing and Information Science in Engineering, 1, pp.235-244. 2001.

Hansen, A. and F. Arbab. An algorithm for generating NC tool paths for arbitrarily shaped pockets with islands. ACM Trans. Graph, 11, pp.52-182. 1992.

Hoffman, C.M. and R. Joan-Arinyo. CAD and the product master model. Computer-Aided Design, 30(11), pp.905-918. 1998a.

Hoffman, C. M. and R. Joan-Arinyo. On user-defined features. Computer-Aided Design, 30(5), pp.321-332. 1998b.

Hoffman, C. M. and R. Joan-Arinyo. Distributed maintenance of multiple product views. Computer-Aided Design, 32, pp.421-431. 2000.

Hao, Q., W.M. Shen, Z. Zhang, S.W. Park and J.K. Lee. Agent-based collaborative product design engineering: An industrial case study. Computers in Industry, 57, pp.26-38. 2006.

Imine, A. Flexible concurrency control for real-time collaborative editors. In Proc. of the 28th International conference on distributed computing systems workshops. 2008.

Jha, K. and B. Gurumoorthy. Automatic propagation of feature modification across domains. Computer-Aided Design, 32, pp.691-706. 2000.

Jing, S.X., F.Z. He, S.H. Han, X.T. Cai and H.J. Liu. A method for topological entity correspondence in a replicated collaborative CAD system. Computers in Industry, 60, pp.467-475. 2009.

Kim, K. and G. Elber. A symbolic approach to freeform parametric surface blends. The Journal of Visualization and Computer Animation, 8, pp.69-80. 1997.

Kim, K.Y., Y. Wang, O.S. Muogboh and B.O. Nnaji. Design formalism for collaborative assembly design. Computer-Aided Design, 36, pp.849-871. 2004.

Keyser, J., T. Culver, M. Foskey, S. Krishnan and D. Manocha. ESOLID-a system for exact boundary evaluation. Computer-Aided Design, 36, pp.175-193. 2004.

Kuk, S.H., H.S. Kim, J.K. Lee, S.H. Han and S.W. Park. An e-Engineering Framework based on service-oriented architecture and agent technologies. Computers in Industry, 59, pp.923-935. 2008.

Lamport, L. Time, Clocks, and the Ordering of Events in a Distributed System. Communications of the ACM, 21(7), pp.558-565. 1978.

Laakko, T. and M. Mäntylä. Feature modeling by incremental feature recognition. Computer-Aided Design, 25(8), pp.479-492. 1993.

Leon, J.C. and P. Trompette. A new approach towards free-form surfaces control. Computer Aided Geometric Design, 12, pp.395-416. 1995.

Lee, J.Y. and K. Kim. A feature-based approach to extracting machining features. Computer-Aided Design, 30(13), pp.1019-1035. 1998.

Lee, J.Y., H. Kim and K. Kim. A Web-Enabled Approach to Feature-Based Modeling in a Distributed and Collaborative Design Environment. Concurrent Engineering, 9(1), pp.74-87. 2001.

Liu, X.D. CFACA: component framework for feature-based design and process planning. Computer-Aided Design, 32, pp.397-408. 2000.

Li, W.D., S.K. Ong and A.Y.C. Nee. Recognizing manufacturing features from a design-by-feature model. Computer-Aided Design, 34, pp.849-868. 2001.

Lee, J.Y., J.H. Lee, H. Kim and H.S. Kim. A cellular topology-based approach to generating progressive solid models from feature-centric models. Computer-Aided Design, 36, pp.217-229. 2004.

Li, W.D., S.K. Ong, J.Y.H. Fuh, Y.S. Wong, Y.Q. Lu and A.Y.C. Nee. Feature-based design in a distributed and collaborative environment. Computer-Aided Design, 36, pp.775-797. 2004a.

Li, W.D., J.Y.H. Fuh and Y.S. Wong. An Internet-enabled integrated system for co-design and concurrent engineering. Computers in Industry, 55, pp.87-103. 2004b.

Li, W.D. and Z.M. Qiu. State-of-the-art technologies and methodologies for collaborative product development systems. International Journal of Production Research, 44(13), pp.2525-2559. 2006.

Li, M., S. Gao and C.C.L. Wang. Real-Time Collaborative Design With Heterogeneous CAD Systems Based on Neutral Modeling Commands. Journal of Computing and Information Science in Engineering, 7(2), pp.113-125. 2007.

Li, M., J.Y.H. Fuh, Y.F. Zhang and S.M. Gao. Adaptive granular concurrency control for replicated collaborative feature modeling. In Proc. of the 12th International Conference on Computer Supported Cooperative Work in Design, Xi'an, China, 1, pp.116-122. 2008a.

Li, M., S.M. Gao, J.Y.H. Fuh and Y.F. Zhang. Replicated concurrency control for collaborative feature modeling: A fine granular approach. Computers in Industry, 59, pp.873-881. 2008b.

Langerak, T.R. Freeform feature recognition and manipulation to support shape design, PhD. thesis, Delft University of Technology. 2008.

Langerak, T.R., J.S.M. Vergeest. A Dual Environment for 3D Modeling With User-Defined Freeform Features. J. Comput. Inf. Sci. Eng. 9, 024503(3 pp.). 2009.

Martino, T.D., B. Falcidieno, F. Giannini, S. Hassinger and J. Ovtcharova. Feature-based modeling by integrating design and recognition approaches. Computer-Aided Design, 26(8), pp.446-453. 1994.

Martino, T.D., B. Falcidieno and S. Habinger. Design and engineering process integration through a multiple view intermediate modeler in a distributed object-oriented system environment. Computer-Aided Design, 30(6), pp.437-452. 1998.

Marcheix, D. and G. Pierra. A Survey of the Persistent Naming Problem. In Proc. The 7th ACM Symposium on Solid Modeling and Application, SM'02, June 17-21 2002, Saarbrucken, Germany. pp.13-22.

Mun, D.W., S.H. Han, J.W. Kim and Y.C. Oh. A set of standard modeling commands for the history-based parametric approach. Computer-Aided Design, 35, pp.1171-1179. 2003.

Nyirenda, P.J., W.F. Bronsvoot, T.R. Langerak, Y. Song and J.S.M. Vergeest. A Generic Taxonomy for Defining Freeform Feature Classes. Computer-Aided Design and Applications, 2, pp.497-506. 2005.

Nyirenda, P.J., M. Mulbagal and W.F. Bronsvoort. Definition of Freeform Surface Feature Classes. Computer-Aided Design & Applications, 3(5), pp.665-674. 2006.

Nyirenda, P.J. and W.F. Bronsvoort. Numeric and curve parameters for freeform surface feature models. Computers-Aided Design, 40, pp.839-851. 2008.

Nyirenda, P.J. and W.F. Bronsvoort. A framework for extendable freeform surface feature modeling. Computers in Industry, 60, pp.35-67. 2009.

Open CADCADE™ 3D Modeling Kernel, Open CADCADE Inc. Available: www.opencascade.com [Last accessed: 9 December 2009].

Piegl, L.A. and W. Tiller. The NURBS Book, second edition. Springer, Berlin. 1997.

Piegl, L.A. and W. Tiller. Computing offsets of NURBS curves and surfaces. Computer- Aided Design, 31, pp.147-156. 1999.

Park, S.C. and H. Shin. Polygonal chain intersection. Computers & Graphics, 26, pp.341-350. 2002.

Pernot, J.P., S. Guillet, J.-C. Leon, B. Falcidieno and F. Giannini. Shape tuning in fully free-form deformation features. Journal of Computing and Information Science in Engineering. 5, pp.95-103. 2005.

Pernot, J.P., B. Falcidieno, F. Giannini and J.C. Leon. Incorporating free-form features in aesthetic and engineering product design: State-of-the-art report. Computers in Industry, 59, pp.626-637. 2008.

Pekerman, D., G. Elber and M.S. Kim. Self-intersection detection and elimination in freeform curves and surfaces. Computer-Aided Design, 40, pp.150-159. 2008.

Qin, H. and D. Terzopoulos. D-NURBS: A physics-based Framework for Geometric Design, IEEE Transactions on Visualization and Computer Graphics, 2, pp.85-96. 1996.

Qiu, Z.M., Y.S. Wong, J.Y.H. Fuh, Y.P. Chen, Z.D. Zhou, W.D. Li and Y.Q. Lu. Geometric model simplification for distributed CAD. Computer-Aided Design, 36, pp.809-819. 2004.

Requicha, A.A.G. and H.B. Voelcker. Boolean operations in solid modelling: boundary evaluation and merging algorithms. In Proc. the IEEE, January 1985. pp.30-44.

Roller, D. Design by Features: An Approach to High Level Shape Manipulations. Computers in Industry, 12, pp.185-191. 1989.

Rosenman, M. and F.J. Wang. ADOM: A component agent-based design-oriented model for collaborative design. Research in Engineering Design, 11, pp.193-205. 1999.

Rovenski, V. Geometry of Curves and Surfaces with Maple. Birkhäuser Boston, 2000.

Rahmani, K. and B. Arezoo. Boundary analysis and geometric completion for recognition of interacting machining features. Computer-Aided Design, 38, pp.845-856. 2006.

Renner, G. and V. Weiβ. Exact and approximate computation of B-spline. Computer-Aided Design, 36, pp.351-362. 2004.

Shah, J.J. Assessment of features technology. Computer-Aided Design, 23(5), pp.331-343. 1991.

Sheu, L.C., and J.T. Lin. Representation scheme for defining and operating form features. Computer-Aided Design, 25(6), pp.333-347. 1993.

Shah, J.J. and M. Mäntylä. Parametric and Feature-Based CAD/CAM: Concepts, Techniques, and Applications. Wiley. 1995.

Shen, W.M., D.H. Norrie, J.-P. Barthes. Multi-agent systems for concurrent intelligent design and manufacturing. Taylor & Francis. 2001.

Shyamsundar, N. and R. Gadh. Internet-based collaborative product design with assembly features and virtual design spaces. Computer-Aided Design, 33, pp.637-651. 2001.

Song, Y., J.S.M. Vergeest and W.F. Bronsvoort. Fitting and manipulating freeform shapes using templates. In Proc. The International Conference on Shape Modeling and Applications, June 7-9, 2004, Palazzo Ducale, Genova, Italy. pp. 86-94.

Sundararajan, V. and P.K. Wright. Volumetric feature recognition for machining components with freeform surfaces. Computer-Aided Design, 36, pp.11-25. 2004.

Subramani, S. and B. Gurumoorthy. Maintaining associativity between form feature models. Computer-Aided Design, 37, pp.1319-1334. 2004.

Shen, Y., S.K. Ong and A.Y.C. Nee. A framework for multiple-view product representation using Augmented Reality. In Proc. of the 2006 International Conference on Cyberworlds (CW'06), 2006, Switzerland. pp.157-162.

Seong, J.K., G. Elber and M.S. Kim. Trimming local and global self-intersections in offset curves/surfaces using distance maps. Computer-Aided Design, 38, pp.183-193. 2006.

Sunil, V.B. and S.S. Pande. Automatic recognition of features from freeform surface CAD models. Computer-Aided Design, 40, pp.502-517. 2008.

Sprynski, N., N. Szafran, B. Lacolle and L. Biard. Surface reconstruction via geodesic interpolation. Computer-Aided Design, 40, pp.480-492. 2008.

Shen, W.M., Q. Hao and W.D. Li. Computer supported collaborative design: retrospective and perspective. Computers in Industry, 59, pp.855-862. 2008.

Vida, J., R.R. Martin and T. Varady. A survey of blending methods that use parametric surfaces. Computer-Aided Design, 26, pp.341-365. 1994.

van Elsas, P.A. and J.S.M. Vergeest. Displacement feature modelling for conceptual design. Computer-Aided Design, 30, pp.19-27. 1998.

Vosniakos, G. Investigation of Feature-Based Shape Modelling for mechanical Parts with Free Form Surfaces. International Journal of Advanced Manufacturing Technology, 15, pp.188-199. 1999.

van Den Berg, E. Web-based collaborative modeling with SPIFF. Ph.D Thesis, Delft University of Technology. 2000.

van den Berg, E., W.F. Bronsvoort and J.S.M. Vergeest. Freeform feature modelling: concepts and prospects. Computers in Industry, 49, pp.217-233. 2002.

van den Berg, E., H.A. van der Meiden and W.F. Bronsvoort. Specification of Freeform Features. In Proc. Solid Modeling 03, Eighth ACM Symposium on Solid Modeling and Applications, 16-20 June 2003, Seattle, USA, Elber G, and Shapiro V (eds), ACM Press, New York. pp.56-64.

van den Berg, E., R. Bidarra and W.F. Bronsvoort. Construction of freeform feature models with attachments. In Proc. DETC2004, ASME Design Engineering Technical Conferences, 28 September-2 October, 2004, Salt Lake City, Utah, USA. pp.1-10.

van den Berg, E. and W.F. Bronsvoort. Parameterized, constraint-based wrapping of freeform shapes. Computers & Graphics, 31, pp. 89-99. 2007.

Wu, J.J., T.B. Zhang, X.F. Zhang and J. Zhou. A face based mechanism for naming, recording and retrieving topological entities. Computer-Aided Design, 33(10), pp.687-698. 2001.

Wu, D. and R. Sarma. The incremental editing of faceted models in an integrated design environment. Computer-Aided Design, 36, pp.821-833. 2004.

Wang, Y. and B.O. Nnaji. Geometry-based semantic ID for persistent and interoperable reference in feature-based parametric modeling. Computer-Aided Design, 37(10), pp.1081-1093. 2005.

Wang, Y. and B.O. Nnaji. Document-Driven Design for Distributed CAD Services in Service-Oriented Architecture. Journal of Computing and Information Science in Engineering, 6, pp.127-138. 2006.

Wang, L.H. and A.Y.C. Nee. Collaborative design and planning for digital manufacturing. Springer. 2008.

Whited, B. and J. Rossignac. Relative blending. Computer-Aided Design, 41, pp.456-462.
2009.

Xue, D., H. Takeda and T. Kiriyama. An Intelligent Integrated Interactive CAD – A Preliminary Report. In Proc. Intelligent Computer Aided Design, D.C. Brown, M. Waldron and H. Yoshikawa (eds), North-Holland, 1992. pp.163-187.

Xiao, A., H.J. Choi, R. Kulkarni, J.K. Allen, D. Rosen and F. Mistree. A Web-based Distributed Product Realization Environment. In Proc. ASME 2001 Design Engineering Technical Conference and Computers and Information in Engineering Conference, September 9-12 2001, Pittsburgh, PA, pp.1-13.

Xue, L.Y., K. Zhang and C.Z. Sun. An integrated post-locking, multi-versioning, and transformation scheme for consistency maintenance in real-time group editors. In Proc. of the 5th international symposium on Autonomous Decentralized Systems, 26-28 March, 2001. pp.57-64.

Yang, Y.J., S. Cao, J.H. Yong, H. Zhang, J.C. Paul, J.G. Sun and H.J. Gu. Approximate computation of curves on B-spline surfaces. Computer-Aided Design, 40, pp.223-234. 2004.

Zhou, L.Y. and R. Nagi. Design of Distributed Information Systems for Agile Manufacturing Virtual Enterprises Using CORBA and STEP Standards. Journal of Manufacturing Systems, 21(1), pp.14-31. 2002.

Zhang, J.M., K.W. Chan and I. Gibson. Constrained deformation of freeform surfaces using surface features for interactive design. International Journal of Advanced Manufacturing Technology, 22, pp.54-67. 2003.

Zhou, X.H., Y.J. Qiu, G.R. Hua, H.F. Wang and X.Y. Ruan. A feasible approach to the integration of CAD and CAPP. Computer-Aided Design, 39, pp.324-338. 2007.

1. L. Yang, S.K. Ong, A.Y.C. Nee, A Surface Blending Approach for Displacement Features on Freeform Surfaces, Computer-Aided Design, 43(2011), 57-66.
2. L. Yang, S.K. Ong, A.Y.C. Nee, Coordination for Replicated Collaborative Feature Modeling, International Journal on Interactive Design and Manufacturing, 2010, 4(3), 191-200.
3. L. Yang, S.K. Ong, A.Y.C. Nee, A History-Independent Modeling Approach for Feature-based Design, Computer-Aided Design, in third round review.
4. B.X. Li, L. Yang, S.K. Ong and A.Y.C. Nee, 2009, Towards ontology-based information extraction in distributed manufacturing systems, 4th International Conference on Advanced Research in Virtual and Rapid Prototyping (VR@P), 6-10 October 2009, Polytechnic Institute of Lerria, Leiria, Portugal, 483-488.

Appendix A Programming of the Performance Measurement using the Proposed Modeling Approach

The performance measurement programming of the proposed modeling approach is presented in this appendix. For the best behavior model, it is a Block containing 100 non-intersecting Holes, as shown in Fig. A.1. The intersection face portions of each Hole is stored at its creation step, as the intersection face portions of the No. 32 Hole shown in Fig. A.2. For the worst behavior model, it is a Block containing 20 horizontal Holes and 20 vertical Holes, where the horizontal Hole is larger than the vertical Hole, as shown in Fig. A.3. In Fig. A.4, the intersection face portions of the second vertical Hole are shown. The programming works presented in this appendix are as follows: in A.1, two primitive features are constructed, namely, Block and Hole; in A.2, the ‘add feature’ operation and ‘remove feature’ operation of the best behavior model are presented; in A.3, the two operations of the worst behavior model are presented.

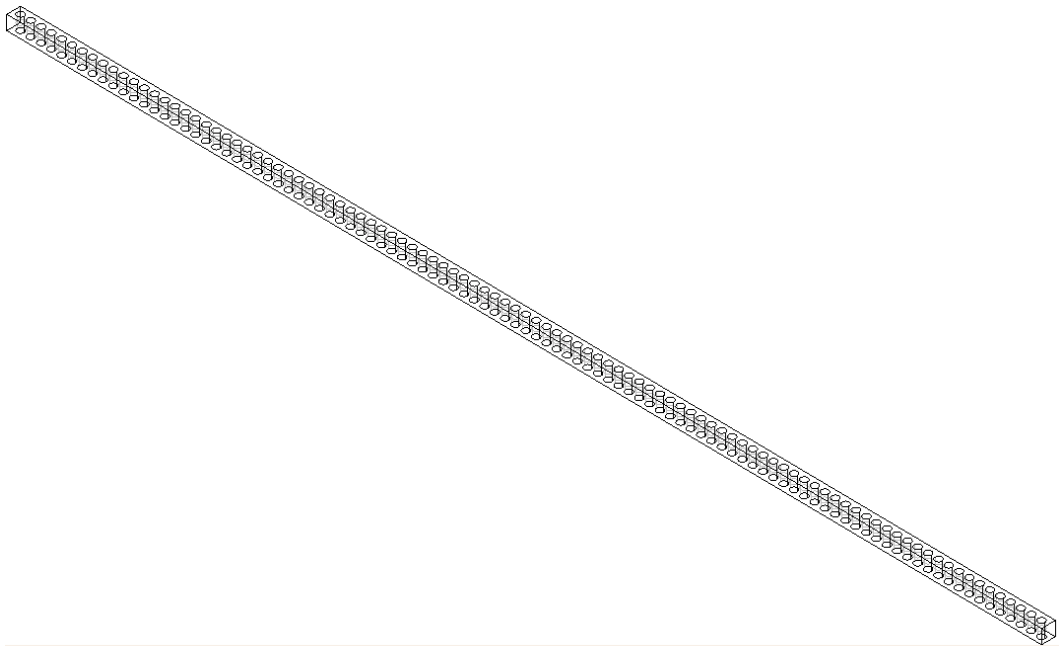


Fig. A.1 The best behavior model

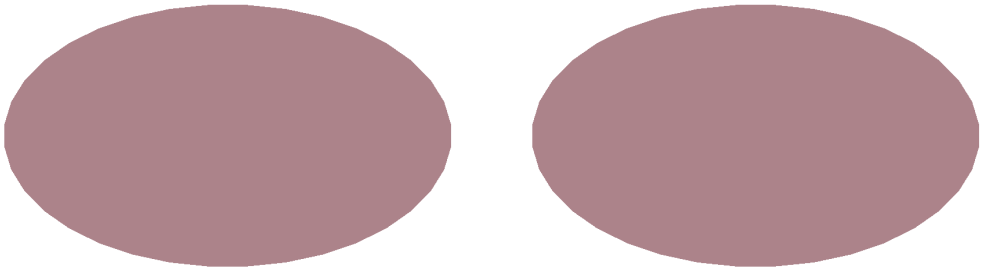


Fig. A.2 Intersection face portions of the 32nd Hole in the best model

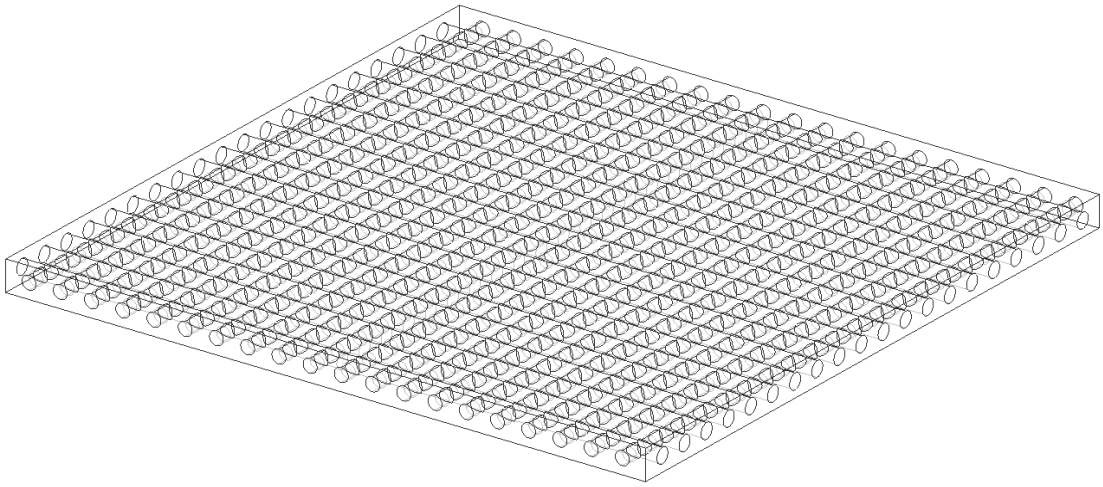


Fig. A.3 The worst behavior model

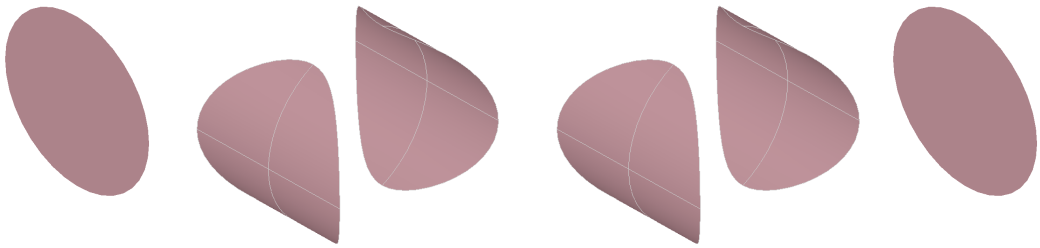


Fig. A.4 Intersection face portions of the second vertical Hole in the worst model

A.1 Primitive Features

```
/***************two primitive features: Block and Hole*****************/
TopoDS_Shape CModelingDoc::Block(Standard_Real x, Standard_Real y,
Standard_Real z, Standard_Real length, Standard_Real width, Standard_Real height)
{
        gp_Pnt point1(x,y,z);
        gp_Pnt point2(x, y+width, z);
        gp_Pnt point3(length+x, y+width, z);
        gp_Pnt point4(length+x, y, z);
        TopoDS_Edge edge1 = BRepBuilderAPI_MakeEdge(point1, point2);
```

```
        TopoDS_Edge edge2 = BRepBuilderAPI_MakeEdge(point2, point3);
        TopoDS_Edge edge3 = BRepBuilderAPI_MakeEdge(point3, point4);
        TopoDS_Edge edge4 = BRepBuilderAPI_MakeEdge(point4, point1);
        TopoDS_Wire sketch = BRepBuilderAPI_MakeWire(edge1, edge2, edge3,
        edge4);
        TopoDS_Face start = BRepBuilderAPI_MakeFace(sketch);
        gp_Vec vec(0, 0, height);
        TopoDS_Shape shape = BRepPrimAPI_MakePrism(start, vec);
        return shape;
}

TopoDS_Shape CModelingDoc::Hole(Standard_Real x, Standard_Real y,
Standard_Real z, Standard_Real radius, Standard_Real depth)
{
        gp_Pnt origin(x, y, z);
        gp_Dir dir(0, 0, 1);
        gp_Ax2 asix(origin, dir);
        gp_Circ circle(asix, radius);
        TopoDS_Edge cirEdge = BRepBuilderAPI_MakeEdge(circle);
        TopoDS_Wire cirWire = BRepBuilderAPI_MakeWire(cirEdge);
        TopoDS_Face cirFace = BRepBuilderAPI_MakeFace(cirWire);
        gp_Vec vec(0, 0, depth);
        TopoDS_Shape shape = BRepPrimAPI_MakePrism(cirFace, vec);
        return shape;
}
/********************end of constructing the primitive features*************/
```

A.2 Measurement of Best Behavior Model

```
/********************measurement of the best behavior model*************/
/************removing the 33rd hole in the model*****************************/
void CModelingDoc::OnMeasureBest()
{
        TopTools_ListOfShape listOfShape;
        BRepTools_ReShape reShape;
        TopoDS_Shape removeFace, myBestModel, result,tempShape;
        clock_t start, finish;
        TopoDS_Shape stock = Block(0,0,0,1505,20,20);
        int n=0,m=0, i = 32;
        CString temp,temp1;
        Handle(TopTools_HArray1OfShape) myTemp = new
                TopTools_HArray1OfShape(0,3);
        ofstream output;
        output.open("C:\\Documents and Settings\\g0501018\\Desktop\\sxlyyl\\
                Myproject\\bestTimeAdd33.txt");
        result = stock;
        for (m=0;m<i;m++)
        {
```

```
            TopoDS_Shape hole = Hole(10+15*m,10,0,5,20);
            result = BRepAlgoAPI_Cut(result, hole);
}
TopoDS_Shape hole;
for(int j=0;j<100;j++)/*measure 100 times to get the average value*/
{
            hole = Hole(10+15*i,10,0,5,20);
            start=clock();
            result = BRepAlgoAPI_Cut(result, hole);
            finish = clock();
            double time =double(finish-start)/CLOCKS_PER_SEC;
            output<<time<<endl;
}
 /********find and store the intersection face portions*****************/
 TopExp_Explorer exp(hole, TopAbs_FACE);
 m=0;
 while(exp.More())
 {
            temp.Format("%d",j);
            temp1.Format("%d", ++m);
            CString path = "C:\\Documents and
            Settings\\Administrator\\Desktop\\Temp\\bestcase\\H"+temp+"Int"+tem
            p1+".brep";
            char* path1 = new char[path.GetLength()+1];
            strcpy(path1,(const char*)path.GetBuffer(0));
            BRepTools::Write(exp.Current(), path1);
            delete path1;
            path.ReleaseBuffer();
            exp.Next();
}
 /********end of find and store the intersection face port*************/
/********for remove and modify operation************************/
for(int n=95;n<100;n++)/**********remove the Nth hole******/
{
            temp.Format("%d",n);
            Standard_CString path = "bestcase/bestmodel.brep";
            BRep_Builder bb;
            BRepTools::Read(myBestModel,path,bb);
            TopExp_Explorer exp2(myBestModel, TopAbs_SHELL);
            TopoDS_Shell modelShell=TopoDS::Shell(exp2.Current());
            /*find the face originating from the hole being removed*/
            TopoDS_Shape hole = Hole(10+15*n,10,0,5,20);
            TopoDS_Shell holeShell =
            BRepTools::OuterShell(TopoDS::Solid(hole));
            BRepAlgoAPI_Common common(holeShell, modelShell);
            TopExp_Explorer exp1(modelShell, TopAbs_FACE);
            while(exp1.More())
            {
                        listOfShape = common.Modified(exp1.Current());
                        if(listOfShape.Extent())
```

```
                        {
                                removeFace = exp1.Current();
                        }
                        exp1.Next();
                }
                /*end of find the face originating from the hole being removed*/
                TopoDS_Shape interFace1,interFace2;
                CString tempStr1 = CString("bestcase/H")+temp+CString("Int2.brep");
                CString tempStr2 = CString("bestcase/H")+temp+CString("Int3.brep");
                char* path1 = new char[tempStr1.GetLength()+1];
                strcpy(path1,(const char*)tempStr2.GetBuffer(0));
                char* path2 = new char[tempStr2.GetLength()+1];
                strcpy(path2,(const char*)tempStr1.GetBuffer(0));
                BRepTools::Read(interFace1,path1,bb);
                BRepTools::Read(interFace2,path2,bb);
                delete path1;
                tempStr1.ReleaseBuffer();
                delete path2;
                tempStr2.ReleaseBuffer();
                /**************measuring time******************/
                ofstream output;
                output.open("C:\\Documents and Settings\\g0501018\\Desktop\\sxlyyl\\
                                Myproject\\bestTimeRemove"+temp+".txt");
                BRepBuilderAPI_Sewing sewing;
                for(int j=0;j<100;j++)/*measure the evaluation time for 100 times*/
                {
                        start=clock();
                        reShape.Remove(removeFace);
                        sewing.Add(interFace1);
                        sewing.Add(interFace2);
                        sewing.Perform();
                        myBestModel= sewing.SewedShape();
                        finish = clock();
                        double time =double(finish-start)/CLOCKS_PER_SEC;
                        output<<time<<endl;
                }/*end of measure 100 times*/
        }
        /************end of remove and modify operation***************/
}
/**********************end of measure the best behavior model***************/
```

A.3 Measurement of Worst Behavior Model

```
/*******************measurement of the worst behavior model**************/
void CModelingDoc::OnMeasureWorst()
{
        clock_t start, finish;
        ofstream output;
```

```
output.open("C:\\Documents and Settings\\g0501018\\Desktop\\sxlyyl\\
                Myproject\\worstTimeRemove10.txt");
TopTools_ListOfShape listOfShape,listOfShape1,listOfShape2;
BRepTools_ReShape reShape;
BRep_Builder bb;
Handle(TopTools_HArray1OfShape) myTemp = new
            TopTools_HArray1OfShape(0,3);
Handle(TopTools_HArray1OfShape) myHoleH = new
            TopTools_HArray1OfShape(0,20);
Handle(TopTools_HArray1OfShape) myHoleV = new
            TopTools_HArray1OfShape(0,20);
Handle(TopTools_HArray1OfShape) myIntFace = new
            TopTools_HArray1OfShape(0,100);
TopoDS_Shape stock = Block(0,0,0,410,410,20);
TopoDS_Shape result,tempShape,hole;
result = stock;
TopoDS_Shell modelShell;
int n=0,m=0,k,i;
for(i=0;i<20;i++)
{
        tempShape = Hole(0,15+20*i,10,5,410);
        myHoleH->SetValue(i,tempShape);
        tempShape = Hole2(15+20*i,0,10,4.5,410);
        myHoleV->SetValue(i,tempShape);
}
i=9; /*add the 9th horizontal and vertical hole**********/
/**************add the horizontal hole******************/
tempShape = myHoleH->Value(i);
for(TopExp_Explorer exp1(tempShape, TopAbs_FACE); exp1.More();
exp1.Next()) {    myTemp->SetValue(m++,exp1.Current());        }
BRepAlgoAPI_Cut cut1(result,tempShape);
for(TopExp_Explorer exp(result, TopAbs_FACE);exp.More();exp.Next())
{
        listOfShape = cut1.Modified(exp.Current());
        if(listOfShape.Extent())/*find and store the intersecting faces*/
        {
                myArrayTempFace->SetValue(n++,exp.Current());
        }
}
for(k=0;k<n;k++)/*remove intersecting faces in the model*/
{
        reShape.Remove(myArrayTempFace->Value(k));
        myIntFace->SetValue(k,BRepAlgoAPI_Cut(myArrayTempFace->
                                        Value(k),tempShape));
}
result= reShape.Apply(result);
BRepBuilderAPI_Sewing sewing;
sewing.Add(result);
for(k=0;k<n;k++)/*add the updated intersecting faces to the model*/
{
```

```
        sewing.Add(myIntFace->Value(k));
}
if(i==0)/*add the new faces in the new feature to the model*/
        sewing.Add(myTemp->Value(0));
else
        {
                TopoDS_Shape tempS2 = myTemp->Value(0);
                for(k=0;k<i;k++)
                {
                tempS2 = BRepAlgoAPI_Cut (tempS2,myHoleV->Value(k));
                }
                sewing.Add(tempS2);
         }
sewing.Perform();
m=0;
n=0;
/**********for add vertical hole***************/
result = sewing.SewedShape();
tempShape = myHoleV->Value(i);
for(TopExp_Explorer
exp2(tempShape,TopAbs_FACE);exp2.More();exp2.Next())
        myTemp->SetValue(m++,exp2.Current());
start = clock();
BRepAlgoAPI_Cut cut2(result,tempShape);
CString temp;
for(TopExp_Explorer exp3(result, TopAbs_FACE);exp3.More();exp3.Next())
{
        listOfShape = cut2.Modified(exp3.Current());
        temp.Format("%d",n+1);
        CString tempCS =
        CString("worstcase/H4")+CString("Int")+temp+".brep";
        char* path = new char[tempCS.GetLength()+1];
        strcpy(path,(const char*)tempCS.GetBuffer(0));
        if(listOfShape.Extent())/*find and store the intersecting faces*/
        {
                myArrayTempFace->SetValue(n++,exp3.Current());
                BRepTools::Write(BRepAlgoAPI_Common(exp3.Current(),tem
                pShape),path);
        }
        delete path;
        tempCS.ReleaseBuffer();
}
for(k=0;k<n;k++)
{
        reShape.Remove(myArrayTempFace->Value(k));/*remove inter face*/
        myIntFace->SetValue(k,BRepAlgoAPI_Cut(myArrayTempFace-
                        >Value(k),tempShape));
}
result= reShape.Apply(result);
BRepBuilderAPI_Sewing sewing2;
```

```
        sewing2.Add(result);
        for(k=0;k<n;k++)
        {
                sewing2.Add(myIntFace->Value(k));/*add the update inter face*/
        }
        if(i==0) /*add the face from the vertical hole being added*/
        sewing2.Add(BRepAlgoAPI_Cut(myTemp->Value(0),myHoleH->Value(0)));
        else /*add the face from the vertical hole being added*/
        {
                TopoDS_Shape tempS2 = myTemp->Value(0);
                int a=0;
                for(k=0;k<i+1;k++)
                {
                        for(TopExp_Explorer exp(myHoleH->Value(k),
                        TopAbs_FACE); exp.More();exp.Next())
                                myTemp->SetValue(a++,exp.Current());

                        BRepAlgoAPI_Fuse fuse (tempS2,myTemp->Value(0));
                                TopTools_ListOfShape listShape;
                        listShape = fuse.Modified(tempS2);
                        sewing2.Add(listShape.First());
                        tempS2 = listShape.Last();
                        a=0;
                }
                sewing2.Add(tempS2);
        }
        sewing2.Perform();
        result = sewing2.SewedShape();
        finish = clock();
        double time =double(finish-start)/CLOCKS_PER_SEC;
        output<<time<<endl;
/**********************end of add operation**********************/

/********************begin of remove operation*********************/
        i=19;//remove the 19th vertical hole
        int count=0;
        for(int j=0;j<10;j++)//measure 10 times to get the average value//
        {
                BRepBuilderAPI_Sewing sewing;
                n=0;
                Standard_CString path1 = "worstcase/model.brep";
                BRepTools::Read(result,path1, bb);
                tempShape = myHoleV->Value(i);
                for(TopExp_Explorer exp1(tempShape, TopAbs_FACE);
                exp1.More();exp1.Next())
                myTemp->SetValue(m++,exp1.Current());
                m=0;
                //finding the faces originating from the hole being removed//
                BRepAlgoAPI_Common common(tempShape,result);
```

```
for(TopExp_Explorer exp(result,TopAbs_FAC);exp.More();exp.Next())
{
	count++;
	listOfShape = common.Modified(exp.Current());

	if(listOfShape.Extent())
	{
		myArrayTempFace->SetValue(n++,exp.Current());
	}
}
//end of finding the faces originating from the hole being removed//
CString temp,temp1;
TopoDS_Shape tempIntface;

for(k=1;k<i+4;k++)
{
	temp.Format("%d",k);
	temp1.Format("%d",i+1);
	CString tempCS =
	CString("worstcase/NewH")+temp1+CString("Int")+temp+".bre
	p";
	char* path = new char[tempCS.GetLength()+1];
	strcpy(path,(const char*)tempCS.GetBuffer(0));
	BRepTools::Read(tempIntface,path, bb);
	myArrayRemoveFace->SetValue(k,tempIntface);
	delete path;
	temp.ReleaseBuffer();
}
//retrieve the intersection face portions stored at the step//
start = clock();
//removing face originating from the hole being removed
for(k=1;k<n;k++)
{
	reShape.Remove(myArrayTempFace->Value(k));
}
reShape.Remove(myArrayTempFace->Value(0));
result= reShape.Apply(result);

sewing.Add(result);
//end of removing face originating from the hole being removed
//merge the intersection face portions
for(k=1;k<i+4;k++)
{
	sewing.Add(myArrayRemoveFace->Value(k));
}
//end of merge the intersection face portions
//update the intersecting feature
for(k=i+1;k<20;k++)
{
```

```
                    BRepAlgoAPI_Common
                    common(BRepTools::OuterShell(TopoDS::Solid(myHoleH-
                    >Value(k))),tempShape);
                    sewing.Add(common.Shape());
                }
                //end of update the intersecting feature
                sewing.Perform();
                result = sewing.SewedShape();
                finish = clock();
                double time =double(finish-start)/CLOCKS_PER_SEC;
                output<<time<<endl;
                //display the result
        }
        myWorstModel = result;
/**********************end of remove operation***************************/
}
```

Appendix B Implementation of Example#2 in Chapter 5.3.4

The implementation of the proposed surface blending is conducted using Maple and VC++. In Maple, as presented in B.1, the derivative vectors of the base surface and the derivative vectors of the boundary curve are computed to determine the offset vectors, which are used to offset the sample points in the boundary curve. In VC++, as presented in B.2, the offset 'sample points' obtained in Maple are interpolated as a B-spine curve, and the four types of curves, namely boundary curve, offset 'boundary curve', displaced curve, offset 'displaced curve', are transformed to be compatible for surface blending. For eliminating self-intersection in the offset curve, the intersection is removed in the offset polygon first. Next, the remaining offset points in the offset polygon are re-interpolated as a B-spline curve, which is then re-sampled using the original sampling number.

B.1 Calculation in Maple

$$Bernstein := (i, n, u) \rightarrow \text{binomial}(n, i) \cdot u^{i} \cdot (1 - u)^{(n - i)};$$

```
/**************Construction of the base surface**********************/
p00 := [0, 0, 0]; p10 := [3, 0, 2]; p20 := [6, 0, 2]; p30 := [9, 0, 0]; p01 := [0, 3, 2];
p11 := [3, 3, 4]; p21 := [6, 3, 4]; p31 := [9, 3, 2]; p02 := [0, 6, 0]; p12 := [3, 6, 2];
p22 := [6, 6, 2]; p32 := [9, 6, 0];
P2 := [p00, p10, p20, p30, p01, p11, p21, p31, p02, p12, p22, p32];
```

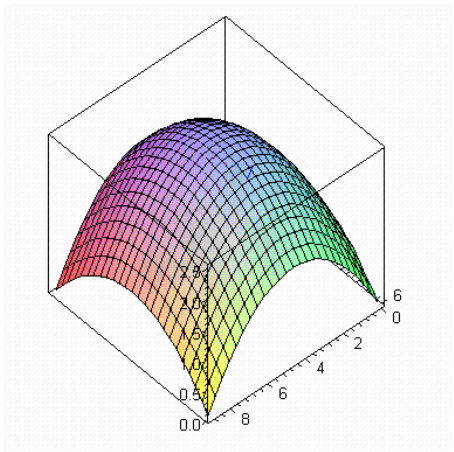
$$surf := (u, v) \rightarrow evalm(sum((sum(Bernstein(i, 3, u) \cdot P2[i + 1 + j \cdot 4], i = 0..3)) \cdot Bernstein(j, 2, v), j = 0..2)) :$$

```
s := plot3d([surf(u, v)[1], surf(u, v)[2], surf(u, v)[3]], u = 0 .. 1, v = 0 .. 1);
/***************end of construction of the base surface***************/
/******the 8 Bezier segments of the feature boundary in the parametric domain****/
P1 := [[.5, .69999999999999996], [.47884017662539796, .69339121717578978],
[.45889469798551014, .68228130340680704],
[.43999999999999995, .66999999999999993]];#arc1
[#arc2-#arc8 are omitted]
/***********determine the 3D space curve of the Bezier segments***********/
```

$$bez1 := t \rightarrow evalm(sum(Bernstein(i, 3, t) \cdot P1[i + 1], i = 0..3)) : c1 := plot([bez1(t)[1], bez1(t)[2], t = 0..1], color = blue) : plots[display](c1) :$$

$$c1\sim := t \rightarrow \mathrm{eval}(\mathrm{subs}(u = \mathrm{bez1}(t)[1], v = bez1(t)[2], surf(u, v))) : m1 := plots[spacecurve]([c1\sim(t)[1], c1\sim(t)[2], c1\sim(t)[3]], t = 0..1, color = blue) :$$

```
plots[display](m1, s, axes = boxed);
```



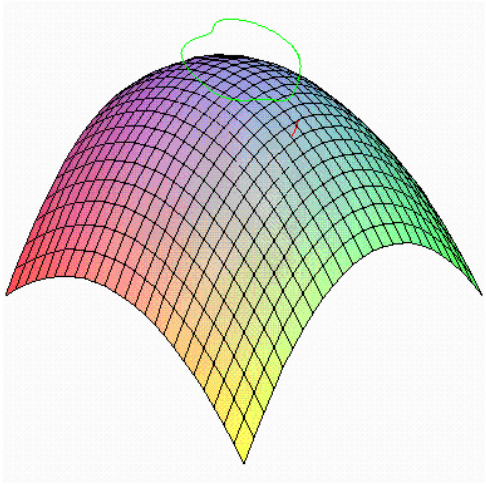

```
/***********determine the control points of the 3D boundary curve**********/
X1 := [4.5, 4.461912318, 4.424136896, 4.386670499, 4.349509891, 4.312651837,
4.276093102, 4.239830451, 4.203860647, 4.168180456, 4.132786643, 4.097675971,
4.062845207, 4.028291114, 3.994010456, 3.96]; Y1 := [4.2, 4.192069461,
4.183367299, 4.173937424, 4.163823743, 4.153070166, 4.141720601, 4.129818957,
4.117409142, 4.104535066, 4.091240637, 4.077569764, 4.063566355, 4.049274318,
4.034737564, 4.02]; Z1 := [2.34, 2.34211481, 2.344190147, 2.346215692,
2.348180833, 2.350074779, 2.35188668, 2.353605729, 2.355221255, 2.356722815,
2.358100265, 2.359343835, 2.360444185, 2.361392459, 2.362180322, 2.3628];
```

$$C1 := t \rightarrow evalm(sum(Bernstein(\mathrm{i}, 15, \mathrm{t}) \cdot [X1[i+1], Y1[i+1], Z1[i+1]], i = 0..15)) : M1 := plots[spacecurve]([C1(t)[1], \mathrm{C1}(t)[2], C1(t)[3]], \mathrm{t} = 0..1, \mathrm{color} = \mathrm{red}) :$$

```
#segment 1
[#segment 2 - #segment 8 omitted]
/****displace the boundary curve towards the exterior of the base surface********/
for d to 16 do CP1[d] := Vector[row]([X1[d], Y1[d], Z1[d]+.5]) end do; #segment1
```

$$DisC1 := t \rightarrow evalm(sum(Bernstein(\mathrm{i}, 15, \mathrm{t}) \cdot CP1[i+1], i = 0..15)) : DisM1 := plots[spacecurve]([DisC1(t)[1], DisC1(t)[2], DisC1(t)[3]], \mathrm{t} = 0..1, \mathrm{color} = \mathrm{green}) :$$

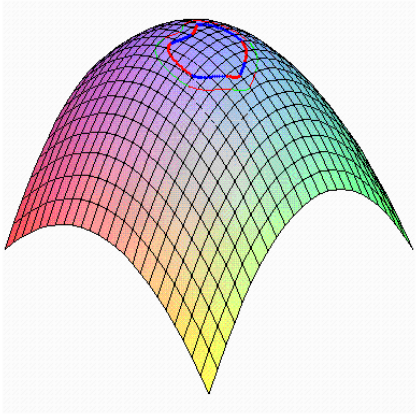
```
plots[display](M1, DisM1, DisM2, DisM3, DisM4, DisM5, DisM6, DisM7, DisM8, s);
```



```
/***************determine the offset vector of the boundary curve*******/
s := [surf(u, v)[1], surf(u, v)[2], surf(u, v)[3]]; Su := diff(s, u); Sv := diff(s, v); Susub :=
eval(subs(u = bez1(t)[1], v = bez1(t)[2], Su)); Svsub := eval(subs(u = bez1(t)[1], v =
bez1(t)[2], Sv)); nt := linalg[crossprod](Susub, Svsub);

BouC := [c1(t)[1], c1(t)[2], c1(t)[3]]; Ct := diff(BouC, t); TT := linalg[crossprod](Ct,
nt); Ctt := diff(Ct, t); Cttt := diff(Ctt, t); b := linalg[crossprod](Ct, Ctt); B := [b[1], b[2],
b[3]]/linalg[norm]([b[1], b[2], b[3]], 2);
/*******determine the sampling number of the offset curve**************/
k := linalg[norm]([b[1], b[2], b[3]], 2)/linalg[norm](Ct, 2)^3; k1 := linalg[norm](Ct,
2)^2*linalg[dotprod](linalg[crossprod](Ct, Cttt), B); k2 := 3*(linalg[dotprod](Ct,
Ctt))(linalg[dotprod](linalg[crossprod](Ct, Ctt), B)); kk := (k1-k2)/linalg[norm](Ct,
2)^5;
CC := (1-.3*k)*Ctt-.3*kk*Ct; for d from 0 to 31 do linalg[norm](evalm(subs(t =
(1/31)*d, CC)), 2) end do;
/*************the offset vector of the sample points*********************/
for d from 0 to 15 do V1[d+1] := Vector[row]([subs(t = (1/15)*d, TT[1]), subs(t =
(1/15)*d, TT[2]), subs(t = (1/15)*d, TT[3])]); V1[d+1] :=
V1[d+1]/linalg[norm](V1[d+1], 2) end do; #segment 1
/******************offset the sample points*************************/
for d from 0 to 15 do CP12[d+1] := evalm(subs(t = (1/15)*d, BouC)-.3*V1[d+1]) end
do;
CP11[1] := Vector[row](3, {(1) = 4.563197514, (2) = 3.916634287, (3) =
2.415564190}); CP11[2] := Vector[row](3, {(1) = 4.530640228, (2) = 3.909487693, (3)
= 2.417221598}); CP11[3] := Vector[row](3, {(1) = 4.498114933, (2) = 3.901698707,
(3) = 2.418838533}); CP11[4] := Vector[row](3, {(1) = 4.465616906, (2) =
3.893289079, (3) = 2.420408060}); CP11[5] := Vector[row](3, {(1) = 4.433142511, (2)
= 3.884281157, (3) = 2.421922930}); CP11[6] := Vector[row](3, {(1) = 4.400689032,
(2) = 3.874697942, (3) = 2.423375632}); CP11[7] := Vector[row](3, {(1) =
4.368254513, (2) = 3.864563128, (3) = 2.424758433}); CP11[8] := Vector[row](3, {(1)
= 4.335837603, (2) = 3.853901119, (3) = 2.426063436}); CP11[9] := Vector[row](3,
{(1) = 4.303437422, (2) = 3.842737038, (3) = 2.427282621}); CP11[10] :=
Vector[row](3, {(1) = 4.271053391, (2) = 3.831096712,(3) = 2.428407888});
CP11[11] := Vector[row](3, {(1) = 4.238685128, (2) = 3.819006652,(3) =
2.429431105}); CP11[12] := Vector[row](3, {(1) = 4.206332320, (2) =
3.806494048,(3) = 2.430344148}); CP11[13] := Vector[row](3, {(1) = 4.173994610,
(2) = 3.793586709,(3) = 2.431138937}); CP11[14] := Vector[row](3, {(1) =
```

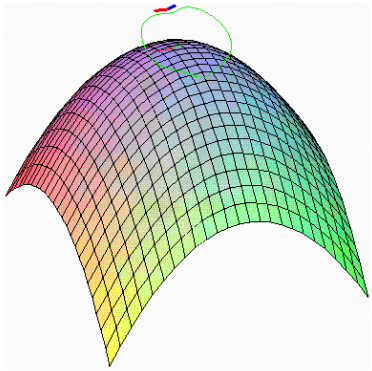
4.141671498, (2) = 3.780313057,(3) = 2.431807478}); CP11[15] := Vector[row](3, {(1) = 4.109362243, (2) = 3.766702071,(3) = 2.432341878}); CP11[16] := Vector[row](3, {(1) = 4.077065779, (2) = 3.752783263,(3) = 2.432734389}); OFF1 := plots[pointplot3d]([seq(CP11[i], i = 1 .. 16)], color = blue); plots[display](M1, M2, M3, M4, M5, M6, M7, M8, OFF2, OFF1, OFF3, OFF4, OFF5, OFF6, OFF7, OFF8, s);



/*********end of offset the sample points*********************/
/**********offset the displaced curve*********************/
OffDisC := [DisC1(t)[1], DisC1(t)[2], DisC1(t)[3]];
for d from 0 to 15 do OffDisCP1[d+1] := evalm(subs(t = (1/15)*d, OffDisC)+.3*V1[d+1]) end do; OFFDis1 := plots[pointplot3d]([seq(OffDisCP1[i], i = 1 .. 16)], color = blue); plots[display](M1, DisM1, OFFDis1);
/*****************remove intersection of curve4 **********/
for d from 0 to 28 do Up[d+1] := subs(t = (1/28)*d, Susub); Vp[d+1] := subs(t = (1/28)*d, Svsub); UpM[d+1] := linalg[norm](Up[d+1], 2); VpM[d+1] := linalg[norm](Vp[d+1], 2); UUp[d+1] := Up[d+1]/UpM[d+1]; UVp[d+1] := Vp[d+1]/VpM[d+1] end do;
/************determine the offset vector in the derivative direction*********/
for d to 29 do VecU[d] := linalg[dotprod](.3*V4[d], UUp[d]); VecV[d] := linalg[dotprod](.3*V4[d], UVp[d]) end do;
/**************determine the offset vector in the domain space********/
for d to 29 do DU[d] := VecU[d]/UpM[d]; DV[d] := VecV[d]/VpM[d] end do;
/*******************offset the domain curve4**************/
for d from 0 to 28 do DP4[d+1] := [subs(t = (1/28)*d, bez1(t)[1]), subs(t = (1/28)*d, bez1(t)[2])]+[DU[d+1], DV[d+1]] end do;
/*****do the same for curve5, and remove the intersection [omitted] **********/
/*get the remaining offset points in curve4 and curve5*/
DP[1] := [.4620217182, .2847979055]; DP[2] := [.4645870586, .2863146607];
DP[3] := [.4670388800, .2878676247]; DP[4] := [.4693866810, .2894413983];
DP[5] := [.4716388264, .2910226484]; DP[6] := [.4738024697, .2925995814];
DP[7] := [.4758835416, .2941615274]; DP[8] := [.4778867749, .2956986262];
DP[9] := [.4798157482, .2972015966]; DP[10] := [.4816729335, .2986615760];
DP[11] := [.4834597399, .3000700196]; DP[12] := [.4851765511, .3014186504];
DP[13] := [.4868227519, .3026994553]; DP[14] := [.4883967479, .3039047230];
DP[15] := [.4898959780, .3050271266]; DP[16] := [.4913169238, .3060598491];
DP[17] := [.4926551245, .3069967616]; DP[18] := [.4939052052, .3078326512];
DP[19] := [.4950609314, .3085635114]; DP[20] := [.4961153090, .3091868913];
DP[21] := [.4970607523, .3097023036]; DP[22] := [.4978893475, .3101116801];
DP[23] := [.4985932465, .3104198438]; DP[24] := [.4991652247, .3106349481];

```
DP[25] := [.4995994290, .3107688016]; DP[26] := [.4998923250, .3108369598];
DP[27] := [.5000084867, .3108359610]; DP[28] := [.5003065830, .3107651677];
DP[29] := [.5007468457, .3106274381]; DP[30] := [.5013255051, .3104073036];
DP[31] := [.5020365187, .3100930982]; DP[32] := [.5028725197, .3096768408];
DP[33] := [.5038255634, .3091538936]; DP[34] := [.5048876777, .3085225090];
DP[35] := [.5060512519, .3077833494]; DP[36] := [.5073092952, .3069390283];
DP[37] := [.5086555995, .3059937002]; DP[38] := [.5100848356, .3049527111];
DP[39] := [.5115926061, .3038223106]; DP[40] := [.5131754719, .3026094246];
DP[41] := [.5148309661, .3013214823]; DP[42] := [.5165576045, .2999662921];
DP[43] := [.5183548988, .2985519643]; DP[44] := [.5202233768, .2970868783];
DP[45] := [.5221646106, .2955796925]; DP[46] := [.5241812565, .2940394026];
DP[47] := [.5262770983, .2924754527]; DP[48] := [.5284570963, .2908979090];
DP[49] := [.5307274273, .2893177059]; DP[50] := [.5330955057, .2877469821];
DP[51] := [.5355699607, .2861995195]; DP[52] := [.5381605398, .2846912994];
/**************determine the parameter values for interpolation********/
for d to 51 do Vpp[d] := Vector[row](DP[d+1]-DP[d]) end do; for d to 51 do
VppM[d] := linalg[norm](Vpp[d], 2) end do; chord := 0.; for d to 51 do chord :=
chord+VppM[d] end do; uu[1] := 0; uu[52] := 1; for d from 2 to 51 do uu[d] := uu[d-
1]+VppM[d-1]/chord end do; for d to 52 do uu[d] end do;
/*************get the newly sample ‘domain points’ after removing intersection****/
DUV := [[.4620217182, .2847979055], [.4634512709, .2856289951],
[.4648653398, .286485992], [.4662647032, .287366857],
[.4676501144, .2882695198], [.4690223923, .2891919916],
[.4703824416, .2901323885], [.4717312155, .2910888874],
[.4730697081, .2920597099], [.4743989695, .2930431305],
[.475720095, .2940374503], [.4770342286, .295040988], [.478342573, .2960520604],
[.4796463896, .2970689626], [.480947011, .2980899468],
[.4822458588, .2991131863], [.4835444557, .3001367437],
[.4848444504, .3011585257], [.4861476543, .3021762127],
[.4874560834, .3031871737], [.4887720174, .3041883471],
[.4900980857, .3051760587], [.4914373945, .3061457398],
[.4927937196, .307091469], [.4941718212, .3080051574],
[.4955779972, .3088749907], [.4970212127, .3096816905],
[.4985158317, .3103878381], [.5000948058, .3108231197]];
/**************get the newly offset vector in domain space************/
for d to 29 do RemVec[d] := [DUV[d][1], DUV[d][2]]+[-subs(t = (d-1)*1/28,
bez1(t)[1]), -subs(t = (d-1)*1/28, bez1(t)[2])] end do;
/**************get the offset displace in derivative direction in base surface***/
for d to 29 do ZU[d] := RemVec[d][1]*Up[d]; ZV[d] := RemVec[d][2]*Vp[d];
ZVec[d] := ZU[d]+ZV[d] end do;
/**************determine the newly offset points SP*************/
for d to 29 do PP1[d] := subs(t = (d-1)*1/28, BouC)+ZU[d]; PP2[d] := subs(t = (d-
1)*1/28, BouC)+ZV[d]; PP3[d] := subs(t = (d-1)*1/28, BouC); PP4[d] :=
Vector[row]([subs(t = (d-1)*1/28, nt[1]), subs(t = (d-1)*1/28, nt[2]), subs(t = (d-
1)*1/28, nt[3])]) end do;
for d to 29 do solve({(x-PP1[d][1])*UUp[d][1]+(y-PP1[d][2])*UUp[d][2]+(z-
PP1[d][3])*UUp[d][3] = 0, (x-PP2[d][1])*UVp[d][1]+(y-PP2[d][2])*UVp[d][2]+(z-
PP2[d][3])*UVp[d][3] = 0, (x-PP3[d][1])*PP4[d][1]+(y-PP3[d][2])*PP4[d][2]+(z-
PP3[d][3])*PP4[d][3] = 0}, {z, x, y}) end do;
```

```
SP41[1] := [4.162299935, 1.707173383, 2.815045962]; SP41[2] := [4.175017126,
1.712262649, 2.817148559]; SP41[3] := [4.187597415, 1.717507035, 2.819277655];
SP41[4] := [4.200047111, 1.722893701, 2.821428307]; SP41[5] := [4.212372356,
1.728409659, 2.823595529]; SP41[6] := [4.224579964, 1.734042451, 2.825774463];
SP41[7] := [4.236677596, 1.739780281, 2.827960375]; SP41[8] := [4.248673425,
1.745611768, 2.830148571]; SP41[9] := [4.260576071, 1.751525838, 2.832334349];
SP41[10] := [4.272394741, 1.757511781, 2.834512980]; SP41[11] := [4.284139123,
1.763559092, 2.836679684]; SP41[12] := [4.295819413, 1.769657419, 2.838829584];
SP41[13] := [4.307446409, 1.775796441, 2.840957704]; SP41[14] := [4.319031500,
1.781965748, 2.843058928]; SP41[15] := [4.330586781, 1.788154722, 2.845127987];
SP41[16] := [4.342125193, 1.794352300, 2.847159436]; SP41[17] := [4.353660652,
1.800546802, 2.849147607]; SP41[18] := [4.365208234, 1.806725647, 2.851086612];
SP41[19] := [4.376784537, 1.812874926, 2.852970262]; SP41[20] := [4.388408015,
1.818978892, 2.854791993]; SP41[21] := [4.400099538, 1.825019254, 2.856544775];
SP41[22] := [4.411883136, 1.830974065, 2.858220925]; SP41[23] := [4.423787161,
1.836816042, 2.859811815]; SP41[24] := [4.435845994, 1.842509823, 2.861307376];
SP41[25] := [4.448102887, 1.848007075, 2.862695184]; SP41[26] := [4.460614914,
1.853237159, 2.863958699]; SP41[27] := [4.473463131, 1.858084716, 2.865072971];
SP41[28] := [4.486777078, 1.862325654, 2.865992473]; SP41[29] := [4.500853252,
1.864938718, 2.866521894];
OFF41 := plots[pointplot3d]([seq(SP41[i], i = 1 .. 29)], color = blue):
plots[display](M4,M5,DisM1, DisM2, DisM3, DisM4, DisM5, DisM6, DisM7,
DisM8,OFF41,OFF51, s);
```



B.2 Surface Construction in VC++

```
void CMapleBlendingDoc::OnMapleDemo1()
{
        /***************base surf, same control points as in B.1***************/
        TColgp_Array2OfPnt Poles(0, 3, 0, 2);
        Handle(Geom_BezierSurface) mySurf = new Geom_BezierSurface(Poles);
        TopoDS_Face Face = BRepBuilderAPI_MakeFace(mySurf);
        /*******************end of base surf*******************************/
        /*********************domain curve*********************************/
        Handle(TColgp_HArray1OfPnt2d) DIntpoints = new
                TColgp_HArray1OfPnt2d(1,8);
        DIntpoints->SetValue(1, gp_Pnt2d(0.5, 0.7));
        DIntpoints->SetValue(2, gp_Pnt2d(0.44, 0.67));
        DIntpoints->SetValue(3, gp_Pnt2d(0.35, 0.55));
```

```
DIntpoints->SetValue(4, gp_Pnt2d(0.45, 0.33));
DIntpoints->SetValue(5, gp_Pnt2d(0.5, 0.36));
DIntpoints->SetValue(6, gp_Pnt2d(0.55, 0.33));
DIntpoints->SetValue(7, gp_Pnt2d(0.65, 0.55));
DIntpoints->SetValue(8, gp_Pnt2d(0.56, 0.67));
Geom2dAPI_Interpolate DInttoBSpline(DIntpoints,1,1.0e-3);
DInttoBSpline.Perform();
Handle(Geom2d_BSplineCurve) DIntBspCur = DInttoBSpline.Curve();
int NbPo = DIntBspCur->NbPoles();
gp_Pnt2d poles[9];
for(int i=0;i<9;i++)
  poles[i] = DIntBspCur->Pole(i+1);
/**************end of domain curve************************/
/*************convert to bspline curve of domain bezier curve********/
Geom2dConvert_BSplineCurveToBezierCurve toBezCur(DIntBspCur);
int NbArc = toBezCur.NbArcs();/*8 arcs*/
Handle(Geom2d_BezierCurve) bezCur = toBezCur.Arc(1);/*do same f other
arcs*/
gp_Pnt2d bezPoles[4];
for(i=0;i< bezCur->NbPoles();i++)
bezPoles[i] = bezCur->Pole(i+1);
/**********end of domain curve convert**********************/
/********input the domain curve in Maple to get 3D curve***********/
/**************boundary curve 1************************/
TColgp_Array1OfPnt cPoles1(1,16);
double Xpoles1[16] = {4.5, 4.461912318, 4.424136896, 4.386670499,
4.349509891, 4.312651837, 4.276093102, 4.239830451, 4.203860647,
4.168180456, 4.132786643, 4.097675971, 4.062845207, 4.028291114,
3.994010456, 3.96};
double Ypoles1[16] = {4.2, 4.192069461, 4.183367299, 4.173937424,
4.163823743, 4.153070166, 4.141720601, 4.129818957, 4.117409142,
4.104535066, 4.091240637, 4.077569764, 4.063566355, 4.049274318,
4.034737564, 4.02};
double Zpoles1[16] = {2.34, 2.34211481, 2.344190147, 2.346215692,
2.348180833, 2.350074779, 2.35188668, 2.353605729, 2.355221255,
2.356722815, 2.358100265, 2.359343835, 2.360444185, 2.361392459,
2.362180322, 2.3628};
for(i=0;i<16;i++)
        cPoles1.SetValue(i+1,gp_Pnt(Xpoles1[i],Ypoles1[i],Zpoles1[i]));
Handle(Geom_BezierCurve) bezCur1 = new Geom_BezierCurve(cPoles1);
/******************convert bezCur1 to Bspline curve*********/
TColStd_Array1OfReal knot(1,2);
knot.SetValue(1,0);
knot.SetValue(2,1);
TColStd_Array1OfInteger multiplicity(1,2);
multiplicity.SetValue(1,16);
multiplicity.SetValue(2,16);
Handle(Geom_BSplineCurve) firstBspCur1 = new
        Geom_BSplineCurve(cPoles1,knot,multiplicity,15);
[do the same program for boundary curve2-curve8]
```

```
/*******get the offset points of the boundary curve from Maple********/
/*********offset curve1, d=0.3, 1.0e-5**********************/
TColgp_Array1OfPnt OFFpoints1(1,16);
OFFpoints1.SetValue(1, gp_Pnt(4.563197514,3.916634287,2.415564190));
OFFpoints1.SetValue(2, gp_Pnt(4.530640228,3.909487693,2.417221598));
OFFpoints1.SetValue(3, gp_Pnt(4.498114933,3.901698707,2.418838533));
OFFpoints1.SetValue(4, gp_Pnt(4.465616906,3.893289079,2.420408060));
OFFpoints1.SetValue(5, gp_Pnt(4.433142511,3.884281157,2.421922930));
OFFpoints1.SetValue(6, gp_Pnt(4.400689032,3.874697942,2.423375632));
OFFpoints1.SetValue(7, gp_Pnt(4.368254513,3.864563128,2.424758433));
OFFpoints1.SetValue(8, gp_Pnt(4.335837603,3.853901119,2.426063436));
OFFpoints1.SetValue(9, gp_Pnt(4.303437422,3.842737038,2.427282621));
OFFpoints1.SetValue(10, gp_Pnt(4.271053391,3.831096712,2.428407888));
OFFpoints1.SetValue(11, gp_Pnt(4.238685128,3.819006652,2.429431105));
OFFpoints1.SetValue(12, gp_Pnt(4.206332320,3.806494048,2.430344148));
OFFpoints1.SetValue(13, gp_Pnt(4.173994610,3.793586709,2.431138937));
OFFpoints1.SetValue(14, gp_Pnt(4.141671498,3.780313057,2.431807478));
OFFpoints1.SetValue(15, gp_Pnt(4.109362243,3.766702071,2.432341878));
OFFpoints1.SetValue(16, gp_Pnt(4.077065779,3.752783263,2.432734389));
/***************interpolate the offset points to Bspline*********/
TColStd_Array1OfReal parameter1(1,16);
for(i=1;i<17;i++)
{
  double para = (i-1)/15.0;
  parameter1.SetValue(i, para);
}
GeomAPI_PointsToBSpline OFFtoBSpline1(OFFpoints1,parameter1);
Handle(Geom_BSplineCurve) offBspCur1 = OFFtoBSpline1.Curve();
[do the same program for boundary curve2-curve8]
/***************displace the boundary curve***********************/
/**************displace curve 1*********************************/
TColgp_Array1OfPnt DiscPoles1(1,16);
for(i=0;i<16;i++)

  DiscPoles1.SetValue(i+1,gp_Pnt(Xpoles1[i],Ypoles1[i],Zpoles1[i]+0.5));
Handle(Geom_BezierCurve) DisbezCur1 = new
Geom_BezierCurve(DiscPoles1);
Handle(Geom_BSplineCurve) DisBspCur1 = new
        Geom_BSplineCurve(DiscPoles1,knot,multiplicity,15);
[do the same program for boundary curve2-curve8]
/***************end of displace boundary curve***********************/
/************************offset displace curve*****************/
/************get the offset points of displaced curve from Maple***/
/************offset displace curve1, d=0.3, 1.0e-5******************/
TColgp_Array1OfPnt DisOFFpoints1(1,16);
DisOFFpoints1.SetValue(1, gp_Pnt(4.436802486,4.483365713,2.764435810));
DisOFFpoints1.SetValue(2, gp_Pnt(4.393474980,4.473942887,2.766968382));
DisOFFpoints1.SetValue(3, gp_Pnt(4.350693135,4.463793125,2.769450857));
DisOFFpoints1.SetValue(4, gp_Pnt(4.308456434,4.452965713,2.771873044));
DisOFFpoints1.SetValue(5, gp_Pnt(4.266763289,4.441509337,2.774224898));
```

```
DisOFFpoints1.SetValue(6, gp_Pnt(4.225611176,4.429472028,2.776496576));
DisOFFpoints1.SetValue(7, gp_Pnt(4.184996817,4.416901128,2.778678489));
DisOFFpoints1.SetValue(8, gp_Pnt(4.144916323,4.403843267,2.780761338));
DisOFFpoints1.SetValue(9, gp_Pnt(4.105365360,4.390344360,2.782736159));
DisOFFpoints1.SetValue(10, gp_Pnt(4.066339257,4.376449614,2.784594348));
DisOFFpoints1.SetValue(11, gp_Pnt(4.027833162,4.362203544,2.786327701));
DisOFFpoints1.SetValue(12, gp_Pnt(3.989842158,4.347650008,2.787928432));
DisOFFpoints1.SetValue(13, gp_Pnt(3.952361366,4.332832219,2.789389207));
DisOFFpoints1.SetValue(14, gp_Pnt(3.915386054,4.317792797,2.790703164));
DisOFFpoints1.SetValue(15, gp_Pnt(3.878911729,4.302573795,2.791863920));
DisOFFpoints1.SetValue(16, gp_Pnt(3.842934221,4.287216737,2.792865611));
 /***************interpolate offset points to Bspline**************/
TColStd_Array1OfReal Disparameter1(1,16);
for(i=1;i<17;i++)
{
  double para = (i-1)/15.0;
  Disparameter1.SetValue(i, para);
}
GeomAPI_PointsToBSpline DisOFFtoBSpline1(DisOFFpoints1,Disparameter1);
Handle(Geom_BSplineCurve) DisoffBspCur1 = DisOFFtoBSpline1.Curve();
[do the same program for boundary curve2-curve8]
/***************end of offset displaced curve*****************/
/******************compatible the four types of curves*************/
/*************compatible curve 7*********************************/
/*get the knot vector*/
offBspCur7->IncreaseDegree(15);
int nb = offBspCur7->NbKnots();
TColStd_Array1OfInteger KnotMul(1,nb);
offBspCur7->Multiplicities(KnotMul);
int nMul=0;
for (i=1;i<=nb;i++)
{
  nMul+= KnotMul.Value(i);
}
TColStd_Array1OfReal KnotSeq(1,nMul);
offBspCur7->KnotSequence(KnotSeq);
double KV[32];
for(i=0;i<32;i++)
{
  KV[i]=KnotSeq.Value(i+1);
}
DisoffBspCur7->IncreaseDegree(15);
int Disnb = DisoffBspCur7->NbKnots();
TColStd_Array1OfInteger DisKnotMul(1,Disnb);
DisoffBspCur7->Multiplicities(DisKnotMul);
int DisnMul=0;
for (i=1;i<=Disnb;i++)
{
  DisnMul+= DisKnotMul.Value(i);
}
```

```
TColStd_Array1OfReal DisKnotSeq(1,DisnMul);
DisoffBspCur7->KnotSequence(DisKnotSeq);
double DisKV[32];
for(i=0;i<32;i++)
{
  DisKV[i]=KnotSeq.Value(i+1);
}
/************end of get the knoe vector**********************/
[do the same job for other curves]
/*get the bezier curve Bez111, first '1' means first boundary curve, second '1'
means first curve of the four types of curves (boundary curve, offset, displaced,
offset displaced), third '1' means first Bezier curve of the boundary curve*/
offBspCur1->IncreaseDegree(15);
DisoffBspCur1->IncreaseDegree(15);
GeomConvert_BSplineCurveToBezierCurve toBezier11(offBspCur1);
int Nbarc11 = toBezier11.NbArcs ();
Handle(Geom_BezierCurve) Bez111 = toBezier11.Arc(1);
GeomConvert_BSplineCurveToBezierCurve toBezier13(DisoffBspCur1);
int Nbarc13 = toBezier13.NbArcs ();
Handle(Geom_BezierCurve) Bez131 = toBezier13.Arc(1);
/**********end of get the bezier curve of the compatible curve***********/
/******************surf construction*****************************/
TColgp_Array1OfPnt Bez111poles(1,16),Bez131poles(1,16);
Bez111->Poles(Bez111poles);
Bez131->Poles(Bez131poles);
TColgp_Array2OfPnt PatchPoles1(0, 15, 0, 3);
for(i=0;i<16;i++)
{
  PatchPoles1.SetValue(i, 0, cPoles1.Value(i+1));
  PatchPoles1.SetValue(i, 1, Bez111poles.Value(i+1));
  PatchPoles1.SetValue(i, 2, Bez131poles.Value(i+1));
  PatchPoles1.SetValue(i, 3, DiscPoles1.Value(i+1));
}
Handle(Geom_BezierSurface) mySurfPatch1 = new
                Geom_BezierSurface(PatchPoles1);
/*end of compatible boundary curve 1*****************************/
 [do the same job for boundary curve2-8:compatible curves and surface
construction]
/****************displace feature construction*******************/
/*displace the modify region*/
int index = 0;
BRepFeat_SplitShape splitter(Face);
TopoDS_Edge myMappedEdge = BRepBuilderAPI_MakeEdge(DIntBspCur,
          mySurf);
BRepLib::BuildCurve3d(myMappedEdge);
splitter.Add(myMappedEdge, Face);
splitter.Build();
TopTools_ListIteratorOfListOfShape iter(splitter.Modified(Face));
Handle(TopTools_HArray1OfShape) m_Face = new
          TopTools_HArray1OfShape(0,1);
```

```
for(;iter.More();iter.Next())
{
  m_Face->SetValue(index,iter.Value());
  index++;
}
gp_Trsf transformation;
transformation.SetTranslation(gp_Vec(0., 0., 0.5));
TopoDS_Shape m_TrsfSrfRegion = m_Face->Value(1);
m_TrsfSrfRegion.Location(TopLoc_Location(transformation));
Handle(AIS_Shape) ais_shape1 = new AIS_Shape(m_Face->Value(0));
Handle(AIS_Shape) ais_shape2 = new AIS_Shape(m_TrsfSrfRegion);
/*************end of displace the modify region**************/
/********remove the self-intersection in curve4 and curve5********/
/*******get the offset points in domain space from Maple********/
Handle(TColgp_HArray1OfPnt2d) DRemIntpoints = new
        TColgp_HArray1OfPnt2d(1,52);
DRemIntpoints->SetValue(1, gp_Pnt2d(.4620217182, .2847979055));
DRemIntpoints->SetValue(2, gp_Pnt2d(.4645870586, .2863146607));
DRemIntpoints->SetValue(3, gp_Pnt2d(.4670388800, .2878676247));
DRemIntpoints->SetValue(4, gp_Pnt2d(.4693866810, .2894413983));
DRemIntpoints->SetValue(5, gp_Pnt2d(.4716388264, .2910226484));
DRemIntpoints->SetValue(6, gp_Pnt2d(.4738024697, .2925995814));
DRemIntpoints->SetValue(7, gp_Pnt2d(.4758835416, .2941615274));
DRemIntpoints->SetValue(8, gp_Pnt2d(.4778867749, .2956986262));
DRemIntpoints->SetValue(9, gp_Pnt2d(.4798157482, .2972015966));
DRemIntpoints->SetValue(10, gp_Pnt2d(.4816729335, .2986615760));
DRemIntpoints->SetValue(11, gp_Pnt2d(.4834597399, .3000700196));
DRemIntpoints->SetValue(12, gp_Pnt2d(.4851765511, .3014186504));
DRemIntpoints->SetValue(13, gp_Pnt2d(.4868227519, .3026994553));
DRemIntpoints->SetValue(14, gp_Pnt2d(.4883967479, .3039047230));
DRemIntpoints->SetValue(15, gp_Pnt2d(.4898959780, .3050271266));
DRemIntpoints->SetValue(16, gp_Pnt2d(.4913169238, .3060598491));
DRemIntpoints->SetValue(17, gp_Pnt2d(.4926551245, .3069967616));
DRemIntpoints->SetValue(18, gp_Pnt2d(.4939052052, .3078326512));
DRemIntpoints->SetValue(19, gp_Pnt2d(.4950609314, .3085635114));
DRemIntpoints->SetValue(20, gp_Pnt2d(.4961153090, .3091868913));
DRemIntpoints->SetValue(21, gp_Pnt2d(.4970607523, .3097023036));
DRemIntpoints->SetValue(22, gp_Pnt2d(.4978893475, .3101116801));
DRemIntpoints->SetValue(23, gp_Pnt2d(.4985932465, .3104198438));
DRemIntpoints->SetValue(24, gp_Pnt2d(.4991652247, .3106349481));
DRemIntpoints->SetValue(25, gp_Pnt2d(.4995994290, .3107688016));
DRemIntpoints->SetValue(26, gp_Pnt2d(.4998923250, .3108369598));
DRemIntpoints->SetValue(27, gp_Pnt2d(.5000084867, .3108359610));
DRemIntpoints->SetValue(28, gp_Pnt2d(.5003065830, .3107651677));
DRemIntpoints->SetValue(29, gp_Pnt2d(.5007468457, .3106274381));
DRemIntpoints->SetValue(30, gp_Pnt2d(.5013255051, .3104073036));
DRemIntpoints->SetValue(31, gp_Pnt2d(.5020365187, .3100930982));
DRemIntpoints->SetValue(32, gp_Pnt2d(.5028725197, .3096768408));
DRemIntpoints->SetValue(33, gp_Pnt2d(.5038255634, .3091538936));
DRemIntpoints->SetValue(34, gp_Pnt2d(.5048876777, .3085225090));
```

```
        DRemIntpoints->SetValue(35, gp_Pnt2d(.5060512519, .3077833494));
        DRemIntpoints->SetValue(36, gp_Pnt2d(.5073092952, .3069390283));
        DRemIntpoints->SetValue(37, gp_Pnt2d(.5086555995, .3059937002));
        DRemIntpoints->SetValue(38, gp_Pnt2d(.5100848356, .3049527111));
        DRemIntpoints->SetValue(39, gp_Pnt2d(.5115926061, .3038223106));
        DRemIntpoints->SetValue(40, gp_Pnt2d(.5131754719, .3026094246));
        DRemIntpoints->SetValue(41, gp_Pnt2d(.5148309661, .3013214823));
        DRemIntpoints->SetValue(42, gp_Pnt2d(.5165576045, .2999662921));
        DRemIntpoints->SetValue(43, gp_Pnt2d(.5183548988, .2985519643));
        DRemIntpoints->SetValue(44, gp_Pnt2d(.5202233768, .2970868783));
        DRemIntpoints->SetValue(45, gp_Pnt2d(.5221646106, .2955796925));
        DRemIntpoints->SetValue(46, gp_Pnt2d(.5241812565, .2940394026));
        DRemIntpoints->SetValue(47, gp_Pnt2d(.5262770983, .2924754527));
        DRemIntpoints->SetValue(48, gp_Pnt2d(.5284570963, .2908979090));
        DRemIntpoints->SetValue(49, gp_Pnt2d(.5307274273, .2893177059));
        DRemIntpoints->SetValue(50, gp_Pnt2d(.5330955057, .2877469821));
        DRemIntpoints->SetValue(51, gp_Pnt2d(.5355699607, .2861995195));
        DRemIntpoints->SetValue(52, gp_Pnt2d(.5381605398, .2846912994));
        /************interpolate the remaining domain offset points***********/
        /*set the respective parameters, determined from Maple*/
        Handle(TColStd_HArray1OfReal) projPara = new
        TColStd_HArray1OfReal(1,52);
        double chord[52] =
        {0,0.03218506726,0.06352856506,0.09405353285,0.1237722992,0.1526865502
        ,0.1807875868,0.2080567566,0.2344660186,0.2599785925,0.2845496780,0.308
        1272480,0.3306528981,0.3520627906,0.3722887194,0.3912593092,0.40890145
        64,0.4251420084,0.4399097926,0.4531380306,0.4647671978,0.4747483395,0.4
        830468132,0.4896463705,0.4945533971,0.4978010960,0.4990556514,0.502364
        5343,0.5073464611,0.5140327260,0.5224278030,0.5325136152,0.5442538447,
        0.5575980408,0.5724853960,0.5888480711,0.6066140527,0.6257095645,0.646
        0610914,0.6675970601,0.6902492575,0.7139540491,0.7386534355,0.76429599
        13,0.7908377206,0.8182428569,0.8464845679,0.8755456087,0.9054188640,0.9
        361077588,0.9676264833,1};
        Geom2dAPI_Interpolate DRemInttoBSpline(DRemIntpoints,projPara,0,1.0e-5);
        DRemInttoBSpline.Perform();
        Handle(Geom2d_BSplineCurve) DRemIntBspCur = DRemInttoBSpline.Curve();
        /************resample the domain curve using original sampling number***/
        gp_Pnt2d DSamplePnt[57];
        double Dpara=0;
        double f = DRemIntBspCur->FirstParameter();
        double l = DRemIntBspCur->LastParameter();
        for(i=0;i<57;i++)
        {
          Dpara = (l-f)*i/56.0;
          DRemIntBspCur->D0(Dpara,DSamplePnt[i]);
        }
}
```